```
MMM         MMM     000000000     NNN         NNN  TTTTTTTTTTTTTTT     000000000     RRRRRRRRRRRR
MMM         MMM     000000000     NNN         NNN  TTTTTTTTTTTTTTT     000000000     RRRRRRRRRRRR
MMM         MMM     000000000     NNN         NNN  TTTTTTTTTTTTTTT     000000000     RRRRRRRRRRRR
MMMMMM   MMMMMM  000         000  NNN         NNN        TTT        000         000  RRR         RRR
MMMMMM   MMMMMM  000         000  NNN         NNN        TTT        000         000  RRR         RRR
MMMMMM   MMMMMM  000         000  NNN         NNN        TTT        000         000  RRR         RRR
MMM   MMM   MMM  000         000  NNNNNN      NNN        TTT        000         000  RRR         RRR
MMM   MMM   MMM  000         000  NNNNNN      NNN        TTT        000         000  RRR         RRR
MMM   MMM   MMM  000         000  NNNNNN      NNN        TTT        000         000  RRR         RRR
MMM         MMM  000         000  NNN   NNN   NNN        TTT        000         000  RRRRRRRRRRRR
MMM         MMM  000         000  NNN   NNN   NNN        TTT        000         000  RRRRRRRRRRRR
MMM         MMM  000         000  NNN   NNN   NNN        TTT        000         000  RRRRRRRRRRRR
MMM         MMM  000         000  NNN      NNNNNN        TTT        000         000  RRR   RRR
MMM         MMM  000         000  NNN      NNNNNN        TTT        000         000  RRR   RRR
MMM         MMM  000         000  NNN      NNNNNN        TTT        000         000  RRR   RRR
MMM         MMM  000         000  NNN         NNN        TTT        000         000  RRR      RRR
MMM         MMM  000         000  NNN         NNN        TTT        000         000  RRR      RRR
MMM         MMM  000         000  NNN         NNN        TTT        000         000  RRR      RRR
MMM         MMM     000000000     NNN         NNN        TTT           000000000     RRR         RRR
MMM         MMM     000000000     NNN         NNN        TTT           000000000     RRR         RRR
MMM         MMM     000000000     NNN         NNN        TTT           000000000     RRR         RRR
```

```
**FILE**ID**MONMAIN





MM      MM    OOOOOO    NN      NN  MM      MM    AAAAAA    IIIIII    NN      NN
MM      MM    OOOOOO    NN      NN  MM      MM    AAAAAA    IIIIII    NN      NN
MMMM  MMMM  OO      OO  NN      NN  MMMM  MMMM  AA      AA      II      NN      NN
MMMM  MMMM  OO      OO  NN      NN  MMMM  MMMM  AA      AA      II      NN      NN
MM  MM  MM  OO      OO  NNNN    NN  MM  MM  MM  AA      AA      II      NNNN    NN
MM  MM  MM  OO      OO  NNNN    NN  MM  MM  MM  AA      AA      II      NNNN    NN
MM      MM  OO      OO  NN  NN  NN  MM      MM  AA      AA      II      NN  NN  NN
MM      MM  OO      OO  NN  NN  NN  MM      MM  AA      AA      II      NN  NN  NN
MM      MM  OO      OO  NN    NNNN  MM      MM  AAAAAAAAAA      II      NN    NNNN
MM      MM  OO      OO  NN    NNNN  MM      MM  AAAAAAAAAA      II      NN    NNNN
MM      MM  OO      OO  NN      NN  MM      MM  AA      AA      II      NN      NN      ....
MM      MM  OO      OO  NN      NN  MM      MM  AA      AA      II      NN      NN      ....
MM      MM    OOOOOO    NN      NN  MM      MM  AA      AA    IIIIII    NN      NN      ....
MM      MM    OOOOOO    NN      NN  MM      MM  AA      AA    IIIIII    NN      NN      ....

LL            IIIIII      SSSSSSSS
LL            IIIIII      SSSSSSSS
LL              II      SS
LL              II      SS
LL              II      SS
LL              II      SS
LL              II        SSSSSS
LL              II        SSSSSS
LL              II              SS
LL              II              SS
LL              II              SS
LL              II              SS
LLLLLLLLLL    IIIIII    SSSSSSSS
LLLLLLLLLL    IIIIII    SSSSSSSS
```

```
              MONMAIN: Procedure        Returns(Fixed Binary(31))
                                        Options(Ident('V04-000'), Main);
      1
   :  1       /*
   :  1       /*****************************************************************************
   :  1       /*                                                                            *
   :  1       /*  COPYRIGHT (c) 1978, 1980, 1982, 1984 BY                                   *
   :  1       /*  DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.                    *
   :  1       /*  ALL RIGHTS RESERVED.                                                      *
   :  1       /*                                                                            *
   :  1       /*  THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED     *
   :  1       /*  ONLY IN  ACCORDANCE WITH  THE  TERMS  OF  SUCH  LICENSE  AND WITH THE     *
   :  1       /*  INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR  ANY  OTHER     *
   :  1       /*  COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY     *
   :  1       /*  OTHER PERSON.  NO TITLE TO AND OWNERSHIP OF  THE  SOFTWARE IS  HEREBY     *
   :  1       /*  TRANSFERRED.                                                              *
   :  1       /*                                                                            *
   :  1       /*  THE INFORMATION IN THIS SOFTWARE IS  SUBJECT TO CHANGE WITHOUT NOTICE     *
   :  1       /*  AND  SHOULD  NOT  BE  CONSTRUED AS  A COMMITMENT BY DIGITAL EQUIPMENT     *
   :  1       /*  CORPORATION.                                                              *
   :  1       /*                                                                            *
   :  1       /*  DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE  OR  RELIABILITY OF ITS     *
   :  1       /*  SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.                   *
   :  1       /*                                                                            *
   :  1       /*                                                                            *
   :  1       /*****************************************************************************
   :  1       /*/
      1
   :  1       /*
   :  1       /*++
   :  1       /* FACILITY:  VAX/VMS MONITOR Utility
   :  1       /*
   :  1       /* ABSTRACT:  MAIN Routine, including command interface.
   :  1       /*
   :  1       /*
   :  1       /* ENVIRONMENT:
   :  1       /*
   :  1       /*              Unprivileged user mode,
   :  1       /*              except for certain collection routines which
   :  1       /*              run in EXEC or KERNEL mode to access system
   :  1       /*              data bases.
   :  1       /*
   :  1       /* AUTHOR:  Thomas L. Cafarella, April, 1981
   :  1       /*
   :  1
```

```
1       /*
1       /* MODIFIED BY:
1       /*
1       /*      V03-018 TLC1089         Thomas L. Cafarella     26-Jul-1984     11:00
1       /*              Accept a space character in time specification.
1       /*
1       /*      V03-017 TLC1087         Thomas L. Cafarella     25-Jul-1984     15:00
1       /*              Default to /ALL when summarizing.
1       /*
1       /*      V03-016 TLC1075         Thomas L. Cafarella     27-Jun-1984     15:00
1       /*              Add stickiness to /INPUT qualifier.
1       /*
1       /*      V03-015 TLC1073         Thomas L. Cafarella     02-May-1984     13:00
1       /*              Make MAX_INP_FILES limit bigger.
1       /*
1       /*      V03-014 PRS1012         Paul R. Senn            23-Mar-1984     14:00
1       /*              Add wildcard capability for MF summary.
1       /*
1       /*      V03-013 TLC1056         Thomas L. Cafarella     23-Mar-1984     13:00
1       /*              Exclude class which is disabled.
1       /*
1       /*      V03-012 PRS1011         Paul R. Senn            29-Feb-1984     14:00
1       /*              add /FLUSH_INTERVAL qualifier
1       /*
1       /*      V03-011 TLC1052         Thomas L. Cafarella     17-Feb-1984     11:00
1       /*              Add multi-file summary capability.
1       /*
1       /*      V03-010 PRS1002         Paul R. Senn            29-Dec-1983     16:00
1       /*              GLOBALDEF VALUE symbols must now be longwords;
1       /*              Use %REPLACE rather than GLOBALDEF VALUE for any equated
1       /*              symbols which are not 4 bytes in length;
1       /*
1       /*      V03-010 PRS1001         Paul R. Senn            27-Dec-1983     16:00
1       /*              Add ALL CLASSES Pseudo-class
1       /*
1       /*      V03-009 TLC1044         Thomas L. Cafarella     24-Aug-1983     13:00
1       /*              Eliminate CLI "NOCOMD" error for comment lines.
1       /*
1       /*      V03-008 SPC0007         Stephen P. Carney       24-Jun-1983     16:00
1       /*              Add EXECUTE subcommand.
1       /*
1       /*      V03-007 TLC1042         Thomas L. Cafarella     19-Jun-1983     15:00
1       /*              Add /ITEM qualifier for homogeneous classes.
1       /*
1       /*      V03-007 TLC1041         Thomas L. Cafarella     16-Jun-1983     15:00
1       /*              Ignore CLI error message when no command on line.
1       /*
1       /*      V03-007 TLC1038         Thomas L. Cafarella     14-Jun-1983     18:00
1       /*              Make default list of classes replace previous list.
1       /*
1       /*      V03-006 TLC1028         Thomas L. Cafarella     14-Apr-1983     16:00
1       /*              Add interactive user interface.
1       /*
1       /*      V03-005 TLC1019         Thomas L. Cafarella     18-Jun-1982     16:00
1       /*              Change CLI$_NEGATED symbol to CLI$_LOCNEG.
1       /*
```

```
 1        /*      V03-004 TLC1012         Thomas L. Cafarella     30-Mar-1982     13:00
 1        /*              Display user's comment string on screen line 5.
 1        /*
 1        /*      V03-004 TLC1011         Thomas L. Cafarella     29-Mar-1982     20:00
 1        /*              Move system service names for SSERROR msg to static storage.
 1        /*
 1        /*      V03-003 TLC1009         Thomas L. Cafarella     29-Mar-1982     01:00
 1        /*              Get current time when other times are converted.
 1        /*
 1        /*      V03-003 TLC1007         Thomas L. Cafarella     28-Mar-1982     19:00
 1        /*              Add checks for maximum sizes of qualifier values.
 1        /*
 1        /*      V03-002 TLC1003         Thomas L. Cafarella     23-Mar-1982     13:00
 1        /*              Fix up module headers.
 1        /*
 1        /*      V03-001 TLC1001         Thomas L. Cafarella     16-Mar-1982     13:00
 1        /*              Add CTRL-W screen refresh support.
 1        /*
 1        /*--
 1        /*/
 1
```

```
        /*
        /*      +-----------------------------------------------------------------------+
        /*      |                                                                       |
        /*      |                            INCLUDE  FILES                             |
        /*      |                                                                       |
        /*      +-----------------------------------------------------------------------+
        /*/

        %INCLUDE        MONDEF;                         /* Monitor utility structure definitions */
        %INCLUDE        $CHFDEF;                        /* Condition handler facility definitions */
        %INCLUDE        $STSDEF;                        /* Status code definitions */

        /*
        /*      +-----------------------------------------------------------------------+
        /*      |                                                                       |
        /*      |                  SYSTEM SERVICE MACRO DEFINITIONS                     |
        /*      |                                                                       |
        /*      +-----------------------------------------------------------------------+
        /*/

```

```
    1      /*
    1      /*      |-----------------------------------------------------------------+
    1      /*      |                                                                 |
    1      /*      |                    COMPILE-TIME CONSTANTS                       |
    1      /*      |                                                                 |
    1      /*      |-----------------------------------------------------------------+
    1      /*/
    1
    1      %REPLACE NOT_SUCCESSFUL         BY '0'B;               /* Failing status bit */
    1      %REPLACE YES                    BY '1'B;               /* For general use */
    1      %REPLACE NO                     BY '0'B;               /* For general use */
    1
```

```
 :  1        /*
 :  1        /*        |--------------------------------------------------------------------------------+
 :  1        /*        |                                                                                |
 :  1        /*        |                         EXTERNAL STORAGE  DEFINITIONS                          |
 :  1        /*        |                                                                                |
 :  1        /*        |--------------------------------------------------------------------------------+
 :  1        /*/
    1
    1        Declare
    1        MON_ERR          ENTRY (ANY VALUE, ANY, ANY) OPTIONS(VARIABLE),
 :  1                                                              /* Routine to log synchronous errors */
    1        SIGNAL_MON_ERR   ENTRY,                               /* Routine to signal MONITOR errors */
    1        DISPLAY_CLEANUP  ENTRY RETURNS(FIXED BINARY(31)),     /* Procedure clean up DISPLAY processing */
    1        SS$_NORMAL       FIXED BINARY(31) GLOBALREF VALUE,    /* System normal return status */
    1        RMS$_EOF         FIXED BINARY(31) GLOBALREF VALUE,    /* RMS end-of-file return status */
    1        RMS$_NMF         FIXED BINARY(31) GLOBALREF VALUE,    /* RMS no-more-files message for wildcard parsing */
    1        RMS$_FNF         FIXED BINARY(31) GLOBALREF VALUE,    /* RMS file-not-found message */
    1        NORMAL           FIXED BINARY(31) GLOBALDEF;          /* MONITOR normal return status */
    1
```

```
      1     Declare
      1     MNR$_ERREXEREA  FIXED BINARY(31) GLOBALREF VALUE,       /* Error message code */
      1     MNR$_ERRRECFIL  FIXED BINARY(31) GLOBALREF VALUE,       /* Error message code */
      1     MNR$_UNEXPERR   FIXED BINARY(31) GLOBALREF VALUE,       /* Error message code */
      1     MNR$_ERRPARSE   FIXED BINARY(31) GLOBALREF VALUE,       /* Error message code */
      1     CLI$_NOCOMD     FIXED BINARY(31) GLOBALREF VALUE;       /* CLI error message code */
      1
```

```
 :  1        /*
 :  1        /*       +-----------------------------------------------------------------------------+
 :  1        /*       |                                                                             |
 :  1        /*       |                       GLOBAL STORAGE DEFINITIONS                            |
 :  1        /*       |                                                                             |
 :  1        /*       +-----------------------------------------------------------------------------+
 :  1        /**/
    1
    1        Declare
    1        COMMAND_FILE             FILE              GLOBALREF;              /* Execute Command File */
    1
    1        Declare
    1        CURR_ERRCODE             FIXED BINARY(31) GLOBALDEF INIT(0),     /* MONITOR error status code currently expected */
    1        FIRST_MON_CMD            BIT(1) ALIGNED GLOBALDEF INIT('0'B),    /* YES => first MONITOR (DCL-level) cmd executing */
    1        PROMPT                   BIT(1) ALIGNED GLOBALDEF INIT('0'B),    /* YES => prompt user for another subcommand */
    1        EXECUTE                  BIT(1) ALIGNED GLOBALDEF INIT('0'B),    /* YES => read another command from the execute file
    1        DISPLAYING               BIT(1) ALIGNED GLOBALDEF INIT('0'B);    /* YES => terminal display output is active */
    1
    1        Declare
    1        CDBPTR                   POINTER GLOBALDEF,                /* Pointer to CDB (Class Descriptor Block) */
    1        MRBPTR                   POINTER GLOBALDEF,                /* Pointer to MRB (Monitor Request Block) */
    1        DEF_MRBPTR               POINTER GLOBALDEF,                /* Pointer to "default" MRB (Monitor Request Block) */
    1        TEMP_MRBPTR              POINTER GLOBALDEF,                /* Pointer to "temporary" MRB (Monitor Request Block) */
    1        CURR_MRBPTR              POINTER GLOBALDEF,                /* Pointer to "current" MRB (Monitor Request Block) */
    1        ACT_MRBPTR               POINTER GLOBALDEF,                /* Pointer to "active" MRB (Monitor Request Block) */
    1        MCAPTR                   POINTER GLOBALDEF,                /* Pointer to MCA (Monitor Communication Area) */
    1        SPTR                     POINTER GLOBALDEF;                /* Pointer to SYI (System Information Area) */
    1
    1        Declare
    1        QUALPTR                  POINTER GLOBALDEF,                /* Pointer to Qualifier Descriptors Block */
    1        DEFPTR                   POINTER GLOBALDEF;                /* Pointer to Qualifier Default Value Descriptors Block */
    1
    1        Declare
    1        1 STAT_TABLE    GLOBALDEF,                                 /* Table of pointers to str descrs for statistic qualifiers
    1          2 STAT_DESC   (0:STATS-1) POINTER;
    1
    1        Declare
    1        1 PROCD_TABLE   GLOBALDEF,                                 /* Table of pointers to str descrs for PROCESSES display qua
    1          2 PROCD_DESC  (0:PROCDISPS-1) POINTER;
    1
```

```
/*
/*      Counted strings for system service names used in the MNR$_SSERROR error message
/*/

Declare
    1 READEF_STR GLOBALDEF,                                         /* Counted string for $READEF */
            2 L             FIXED BINARY(7) INIT(7),               /* Length */
            2 S             CHAR(7) INIT('$READEF'),               /* String */

    1 CLREF_STR GLOBALDEF,                                          /* Counted string for $CLREF */
            2 L             FIXED BINARY(7) INIT(6),               /* Length */
            2 S             CHAR(7) INIT('$CLREF'),                /* String */

    1 SCHDWK_STR GLOBALDEF,                                         /* Counted string for $SCHDWK */
            2 L             FIXED BINARY(7) INIT(7),               /* Length */
            2 S             CHAR(7) INIT('$SCHDWK'),               /* String */

    1 SETIMR_STR GLOBALDEF,                                         /* Counted string for $SETIMR */
            2 L             FIXED BINARY(7) INIT(7),               /* Length */
            2 S             CHAR(7) INIT('$SETIMR'),               /* String */

    1 DCLAST_STR GLOBALDEF,                                         /* Counted string for $DCLAST*/
            2 L             FIXED BINARY(7) INIT(7),               /* Length */
            2 S             CHAR(7) INIT('$DCLAST');               /* String */
```

```
/*
/*      +-----------------------------------------------------------------------+
/*      |                                                                       |
/*      |                            OWN STORAGE                                |
/*      |                                                                       |
/*      +-----------------------------------------------------------------------+
/*/

Declare
CALL                    FIXED BINARY(31),            /* Holds function value (return status) of called routines *
STATUS                  BIT(1)  BASED(ADDR(CALL));   /* Low-order status bit for called routines */

/*
/*      Strings for command qualifiers and parameter. Descriptors
/*      for these strings are defined in the QUAL structure.
/*/

DECLARE

BEG_QUAL_S      CHAR(9) INIT('BEGINNING'),                      /* BEGINNING qualifier string */
END_QUAL_S      CHAR(6) INIT('ENDING'),                         /* ENDING qualifier string */
INT_QUAL_S      CHAR(8) INIT('INTERVAL'),                       /* INTERVAL qualifier string */
FLUSH_QUAL_S    CHAR(14) INIT('FLUSH_INTERVAL'),                /* FLUSH_INTERVAL qualifier string */
VIEW_QUAL_S     CHAR(12) INIT('VIEWING_TIME'),                  /* VIEWING_TIME qualifier string */
INP_QUAL_S      CHAR(5) INIT('INPUT'),                          /* INPUT qualifier string */
DISP_QUAL_S     CHAR(7) INIT('DISPLAY'),                        /* DISPLAY qualifier string */
REC_QUAL_S      CHAR(6) INIT('RECORD'),                         /* RECORD qualifier string */
SUMM_QUAL_S     CHAR(7) INIT('SUMMARY'),                        /* SUMMARY qualifier string */
COMM_QUAL_S     CHAR(7) INIT('COMMENT'),                        /* COMMENT qualifier string */
BY_NODE_QUAL_S  CHAR(7) INIT('BY_NODE'),                        /* BY_NODE qualifier string */
CLASS_PARM_S    CHAR(10) INIT('CLASS_NAME');                    /* CLASS_NAME parameter string */

/*
/*      Strings for class-name qualifiers. Descriptors for these strings are defined in
/*      the QUAL structure.
/*/

DECLARE

ALL_QUAL_S      CHAR(3) INIT('ALL'),                            /* ALL qualifier string */
CUR_QUAL_S      CHAR(7) INIT('CURRENT'),                        /* CURRENT qualifier string */
AVE_QUAL_S      CHAR(7) INIT('AVERAGE'),                        /* AVERAGE qualifier string */
MIN_QUAL_S      CHAR(7) INIT('MINIMUM'),                        /* MINIMUM qualifier string */
MAX_QUAL_S      CHAR(7) INIT('MAXIMUM'),                        /* MAXIMUM qualifier string */
TOPC_QUAL_S     CHAR(6) INIT('TOPCPU'),                         /* TOPCPU qualifier string */
TOPD_QUAL_S     CHAR(6) INIT('TOPDIO'),                         /* TOPDIO qualifier string */
TOPB_QUAL_S     CHAR(6) INIT('TOPBIO'),                         /* TOPBIO qualifier string */
TOPF_QUAL_S     CHAR(8) INIT('TOPFAULT'),                       /* TOPFAULT qualifier string */
CPU_QUAL_S      CHAR(3) INIT('CPU'),                            /* CPU qualifier string */
PCENT_QUAL_S    CHAR(7) INIT('PERCENT'),                        /* PERCENT qualifier string */
ITEM_QUAL_S     CHAR(4) INIT('ITEM');                           /* ITEM qualifier string */


/*
/*      Default file-spec values for qualifiers. Descriptors for these strings are
/*      defined in the DEF structure.
```

```
     1          /*/

     1          DECLARE

     1          REC_DEF_S        CHAR(11) INIT('MONITOR.DAT'),                  /* RECORD qualifier default value */
     1          DISP_DEF_S       CHAR(11) INIT('SYS$OUTPUT:'),                  /* DISPLAY qualifier default value */
     1          SUMM_DEF_S       CHAR(11) INIT('MONITOR.SUM');                  /* SUMMARY qualifier default value */

     1
```

```
      1
      1         DECLARE
      1
      1         01 MRB_DEF,                                                    /* Default Monitor Request Block */
      1
      1             02 beg      bit(64) aligned          init('0'B),           /* Beginning time of request in system time units */
      1             02 end      bit(64) aligned          init('0'B),           /* Ending time of request in system time units */
      1             02 int      fixed binary(31)         init(0),              /* Interval value in seconds */
      1             02 flush    fixed binary(31)         init(0),              /* Flush interval in seconds */
      1             02 vie      fixed binary(31)         init(0),              /* Viewing time for a screen in seconds */
      1             02 inp      pointer                  init(null()),         /* Address of input file descr (0 if input not requested) */
      1             02 dis      pointer,                                       /* Address of display file descr (0 if display not requested
    : 1                                                                        /* (Initialized in MONITOR_INIT routine) */
      1             02 rec      pointer                  init(null()),         /* Address of record file descr (0 if record not requested)
      1             02 sum      pointer                  init(null()),         /* Address of summary file descr (0 if summary not requested
      1             02 com      pointer                  init(null()),         /* Address of comment string descriptor */
      1             02 clc      fixed binary(15),                              /* Count of classes requested (needs no init) */
      1             02 clb      bit(128) aligned         init('0'B),           /* Bit string of requested classes */
      1             02 inf      fixed binary(7),                               /* Count of input files specified (needs no init) */
      1             02 fla,                                                    /* Flags for MRB */
      1                 03 dis  bit(1)                   init(NO),
      1                 03 rec  bit(1)                   init(NO),
      1                 03 sum  bit(1)                   init(NO),
      1                 03 pbk  bit(1)                   init(NO),
      1                 03 ide  bit(1)                   init(NO),
      1                 03 dtf  bit(1)                   init(NO),
      1                 03 icr  bit(1)                   init(NO),
      1                 03 rcr  bit(1)                   init(NO),
      1                 03 dcr  bit(1)                   init(NO),
      1                 03 scr  bit(1)                   init(NO),
      1                 03 alc  bit(1)                   init(NO),
      1                 03 mfs  bit(1)                   init(NO),
      1                 03 byn  bit(1)                   init(NO),
      1                 03 scl  bit(1)                   init(NO),
      1                 03 fil  bit(2)                   init('0'B);
      1
```

```
 :  1        /*
 :  1        /*++
 :  1        /*
 :  1        /* FUNCTIONAL DESCRIPTION:
 :  1        /*
 :  1        /*      MONMAIN
 :  1        /*
 :  1        /*      This routine is the main Monitor routine, entered from DCL.
 :  1        /*      It calls the CLE (Command Language Editor) to parse the
 :  1        /*      MONITOR command line. and creates a Monitor Request Block (MRB)
 :  1        /*      describing the request. The EXECUTE_REQUEST routine is then
 :  1        /*      called to execute the request.
 :  1        /*
 :  1        /* INPUTS:
 :  1        /*
 :  1        /*      None
 :  1        /*
 :  1        /* IMPLICIT INPUTS:
 :  1        /*
 :  1        /*      The MONITOR command line.
 :  1        /*
 :  1        /* OUTPUTS:
 :  1        /*
 :  1        /*      None
 :  1        /*
 :  1        /* ROUTINE VALUE:
 :  1        /*
 :  1        /*      SS$_NORMAL, or failing MONITOR status.
 :  1        /*
 :  1        /*--
 :  1        /*/
    1
 1          ON FINISH;                                                      /* On finish, do nothing */
 1          ON ANYCONDITION                                                 /* On any condition signaled, */
 1              BEGIN;
 2              DECLARE MON_CODE     FIXED BINARY(31),                      /* Monitor message code */
 2                      TEMP         FIXED BINARY(31),                      /* Temporary scratch area */
 2                      ON_FILE      CHAR(100) VARYING,                     /* Holds possible file name string */
 2                      MNR$_FACNO   FIXED BINARY(31) GLOBALREF VALUE,  /* MONITOR facility number */
 2                      SIGNALED_ERR ENTRY (ANY VALUE, ANY VALUE, ANY VALUE, ANY); /* Rtn to set up PUTMSGVEC */
 2
 2              CHF$ARGPTR = ONARGSLIST();                                  /* Get signal array pointer */
 2              STS$VALUE = CHF$SIG_NAME;                                   /* Get code for signaled condition */
 2              UNSPEC(TEMP) = STS$FAC_NO;                                  /* Convert facility no. to binary in TEMP */
 2              IF TEMP = MNR$_FACNO                                        /* If a MONITOR code, re-signal it */
 2                  THEN CALL RESIGNAL();
 2
 2              IF CURR_ERRCODE = MNR$_ERRPARSE &                           /* If expecting a CLU parsing error, */
 2                 CHF$SIG_NAME = CLI$_NOCOMD                               /* AND it's a "No Command on Line", */
 2                  THEN CALL = NORMAL;                                     /*    then set up code so it's ignored */
 2                  ELSE DO;                                                /* Otherwise */
 3                      IF CURR_ERRCODE = MNR$_ERREXEREA                    /* Was it an execute command file read error? */
 3                          THEN DO;
 4                              CLOSE FILE (COMMAND_FILE);                  /*  Yes, close the execute command file */
 4                              EXECUTE = NO;                               /*  Don't do any more input from the file */
 4                              END;
 3                      IF CURR_ERRCODE = 0                                 /* See if an error is currently expected */
 3                          THEN MON_CODE = MNR$_UNEXPERR;                  /* No, set "unexpected" code */
 3                          ELSE MON_CODE = CURR_ERRCODE;                   /* Yes, set currently expected code */
 3                      CALL SIGNALED_ERR(MON_CODE,STS$VALUE,DIM(CHF$SIG_ARG,1),CHF$SIG_ARG); /* Log the error */
 3                      CALL = MON_CODE;                                    /* Set up code for MONITOR request termination */
 3                      END;
 2
 2              CURR_ERRCODE = 0;                                           /* Reset to default MONITOR error code ("unexpected") */
 2              GO TO MON_REQ_TERM;                                         /* ... and go terminate (PL/I does an UNWIND) */
 2              END;
 1
```

```
  1        CALL = MONITOR_INIT();                                        /* Do image-wide initialization */
  1        IF STATUS                                                     /* Continue if status OK */
  1            THEN CALL = MONITOR_CMD();                                /* Analyze and execute first (DCL-level) MONITOR cmd */
  1
  1        MON_REQ_TERM:                                                 /* MONITOR request termination */
  1
  1        /*
  1        /*      We get to this point by one of three routes:
  1        /*
  1        /*              1) A MONITOR request has just terminated successfully or with an error status code; or
  1        /*              2) A MONITOR request has just terminated with an error that was signaled; or
  1        /*              3) The MONITOR_INIT call above terminated with an error status.
  1        /*
  1        /*      In all three cases we want to do the same thing. That is, to loop prompting for more subcommands
  1        /*      as long as the PROMPT indicator is still set to YES. It can be set to NO by an EXIT subcommand,
  1        /*      or as a result of the user's striking CTRL/Z (either in response to the MONITOR> prompt, or while
  1        /*      a MONITOR request is running). In case 3 above, it will always be set to NO. For all cases, the
  1        /*      variable CALL contains the status code of interest and, if an error, the PUTMSG vector (PUTMSGVEC)
  1        /*      has been set up with error message information. STATUS is a synonym for the low-order bit of CALL.
  1        /*
  1        /*      If the EXECUTE indicator is set to YES then NEXT_EXECUTE_COMMAND will be called.  If EXECUTE is set
  1        /*      to NO, then NEXT_COMMAND is called.  NEXT_EXECUTE_COMMAND will retrieve commands from a file intead
  1        /*      of the terminal as done by EXECUTE_COMMAND.
  1        /**/
  1
  1        IF STATUS = NOT_SUCCESSFUL                                    /* If bad status, */
  1            THEN DO;
  2                IF DISPLAYING = YES                                   /* If display output is active, */
  2                    THEN STS$VALUE = DISPLAY_CLEANUP();               /*    then perform cleanup */
  2                CALL SIGNAL_MON_ERR();                                /* Signal MONITOR error */
  2                END;
  1
  1        DO WHILE (PROMPT = YES);                                      /* Main loop to perform subcommands */
  2            IF EXECUTE = YES                                          /* Read from the execute command file? */
  2                THEN CALL = NEXT_EXECUTE_COMMAND();                   /* Yes, execute next subcommand line from the file */
  2                ELSE CALL = NEXT_COMMAND();                           /* No, Read from the terminal and execute next subcommand li
  2            IF STATUS = NOT_SUCCESSFUL THEN DO;                       /* If bad status, */
  3                IF DISPLAYING = YES                                   /* If display output is active, */
  3                    THEN STS$VALUE = DISPLAY_CLEANUP();               /*    then perform cleanup */
  3                CALL SIGNAL_MON_ERR();                                /* Signal MONITOR error, using PUTMSGVEC */
  3                END;
  2        END;                                                          /* End of subcommand loop */
  1
  1
  1        STS$VALUE = CALL;                                             /* Get MONITOR completion status */
  1        STS$INHIB_MSG = YES;                                          /* Inhibit DCL print */
  1        RETURN(STS$VALUE);                                            /* Return to DCL */
  1
```

```
   1      MONITOR_INIT: Procedure Returns(Fixed Binary(31));
   2
 | 2      /*
 | 2      /*      This routine performs general set-up, including
 | 2      /*      setting of the current MRB to default values.
 | 2      /*/
   2
   2
```

```
      2
      2      PROMPT = NO;                               /* Start out assuming no MONITOR> prompting */
      2      EXECUTE = NO;                              /* Start out assuming no EXECUTE file */
      2      FIRST_MON_CMD = YES;                       /* Indicate first MONITOR command (entered at DCL) */
      2      NORMAL = SS$_NORMAL;                       /* Use system's normal status */
      2
    | 2      /*
    | 2      /*     Allocate the two blocks which contain string descriptors for
    | 2      /*     command qualifiers and default qualifier values. Global
    | 2      /*     pointers to the blocks are automatically established.
    | 2      /*/
      2
      2      ALLOCATE QUALIFIER_DESC;
      2      ALLOCATE DEF_DESC;
      2
      2
    | 2      /*
    | 2      /*     Initialize string descriptors for command
    | 2      /*     qualifiers, command parameters, etc.
    | 2      /*/
      2
      2      QUAL$L_BEG = LENGTH(BEG_QUAL_S);           /* Init length longword of descr */
      2      QUAL$A_BEG = ADDR(BEG_QUAL_S);             /* Init address longword of descr */
      2
      2      QUAL$L_END = LENGTH(END_QUAL_S);           /* Init length longword of descr */
      2      QUAL$A_END = ADDR(END_QUAL_S);             /* Init address longword of descr */
      2
      2      QUAL$L_INT = LENGTH(INT_QUAL_S);           /* Init length longword of descr */
      2      QUAL$A_INT = ADDR(INT_QUAL_S);             /* Init address longword of descr */
      2
      2      QUAL$L_FLUSH = LENGTH(FLUSH_QUAL_S);       /* Init length longword of descr */
      2      QUAL$A_FLUSH = ADDR(FLUSH_QUAL_S);         /* Init address longword of descr */
      2
      2      QUAL$L_VIEW = LENGTH(VIEW_QUAL_S);         /* Init length longword of descr */
      2      QUAL$A_VIEW = ADDR(VIEW_QUAL_S);           /* Init address longword of descr */
      2
      2      QUAL$L_BY_NODE = LENGTH(BY_NODE_QUAL_S);   /* Init length longword of descr */
      2      QUAL$A_BY_NODE = ADDR(BY_NODE_QUAL_S);     /* Init address longword of descr */
      2
      2      QUAL$L_INP = LENGTH(INP_QUAL_S);           /* Init length longword of descr */
      2      QUAL$A_INP = ADDR(INP_QUAL_S);             /* Init address longword of descr */
      2
```

```
 2        QUAL$L_DISP = LENGTH(DISP_QUAL_S);            /* Init length longword of descr */
 2        QUAL$A_DISP = ADDR(DISP_QUAL_S);              /* Init address longword of descr */
 2
 2        QUAL$L_REC = LENGTH(REC_QUAL_S);              /* Init length longword of descr */
 2        QUAL$A_REC = ADDR(REC_QUAL_S);                /* Init address longword of descr */
 2
 2        QUAL$L_SUMM = LENGTH(SUMM_QUAL_S);            /* Init length longword of descr */
 2        QUAL$A_SUMM = ADDR(SUMM_QUAL_S);              /* Init address longword of descr */
 2
 2        QUAL$L_COMM = LENGTH(COMM_QUAL_S);            /* Init length longword of descr */
 2        QUAL$A_COMM = ADDR(COMM_QUAL_S);              /* Init address longword of descr */
 2
 2        QUAL$L_BY_NODE = LENGTH(BY_NODE_QUAL_S);      /* Init length longword of descr */
 2        QUAL$A_BY_NODE = ADDR(BY_NODE_QUAL_S);        /* Init address longword of descr */
 2
 2        QUAL$L_CLASS = LENGTH(CLASS_PARM_S);          /* Init length longword of descr */
 2        QUAL$A_CLASS = ADDR(CLASS_PARM_S);            /* Init address longword of descr */
 2
 2        QUAL$L_ALL = LENGTH(ALL_QUAL_S);              /* Init length longword of descr */
 2        QUAL$A_ALL = ADDR(ALL_QUAL_S);                /* Init address longword of descr */
 2
 2        QUAL$L_CUR = LENGTH(CUR_QUAL_S);              /* Init length longword of descr */
 2        QUAL$A_CUR = ADDR(CUR_QUAL_S);                /* Init address longword of descr */
 2
 2        QUAL$L_AVE = LENGTH(AVE_QUAL_S);              /* Init length longword of descr */
 2        QUAL$A_AVE = ADDR(AVE_QUAL_S);                /* Init address longword of descr */
 2
 2        QUAL$L_MIN = LENGTH(MIN_QUAL_S);              /* Init length longword of descr */
 2        QUAL$A_MIN = ADDR(MIN_QUAL_S);                /* Init address longword of descr */
 2
 2        QUAL$L_MAX = LENGTH(MAX_QUAL_S);              /* Init length longword of descr */
 2        QUAL$A_MAX = ADDR(MAX_QUAL_S);                /* Init address longword of descr */
 2
 2        QUAL$L_TOPC = LENGTH(TOPC_QUAL_S);            /* Init length longword of descr */
 2        QUAL$A_TOPC = ADDR(TOPC_QUAL_S);              /* Init address longword of descr */
 2
 2        QUAL$L_TOPD = LENGTH(TOPD_QUAL_S);            /* Init length longword of descr */
 2        QUAL$A_TOPD = ADDR(TOPD_QUAL_S);              /* Init address longword of descr */
 2
 2        QUAL$L_TOPB = LENGTH(TOPB_QUAL_S);            /* Init length longword of descr */
 2        QUAL$A_TOPB = ADDR(TOPB_QUAL_S);              /* Init address longword of descr */
 2
 2        QUAL$L_TOPF = LENGTH(TOPF_QUAL_S);            /* Init length longword of descr */
 2        QUAL$A_TOPF = ADDR(TOPF_QUAL_S);              /* Init address longword of descr */
 2
 2        QUAL$L_CPU = LENGTH(CPU_QUAL_S);              /* Init length longword of descr */
 2        QUAL$A_CPU = ADDR(CPU_QUAL_S);                /* Init address longword of descr */
 2
 2        QUAL$L_PCENT = LENGTH(PCENT_QUAL_S);          /* Init length longword of descr */
 2        QUAL$A_PCENT = ADDR(PCENT_QUAL_S);            /* Init address longword of descr */
 2
 2        QUAL$L_ITEM = LENGTH(ITEM_QUAL_S);            /* Init length longword of descr */
 2        QUAL$A_ITEM = ADDR(ITEM_QUAL_S);              /* Init address longword of descr */
 2
 2        DEF$L_REC = LENGTH(REC_DEF_S);                /* Init length longword of descr */
 2        DEF$A_REC = ADDR(REC_DEF_S);                  /* Init address longword of descr */
```

```
     2
     2          DEF$L_DISP = LENGTH(DISP_DEF_S);                      /* Init length longword of descr */
     2          DEF$A_DISP = ADDR(DISP_DEF_ST);                       /* Init address longword of descr */
     2
     2          DEF$L_SUMM = LENGTH(SUMM_DEF_S);                      /* Init length longword of descr */
     2          DEF$A_SUMM = ADDR(SUMM_DEF_ST);                       /* Init address longword of descr */
     2
     2
```

```
  !   2          /*
  !   2          /*      Initialize STAT_TABLE descriptor pointers.
  !   2          /*/
      2
      2          STAT_DESC(ALL_STAT) = ADDR(QUAL$L_ALL);           /* Init pointer to ALL descr */
      2          STAT_DESC(CUR_STAT) = ADDR(QUAL$L_CUR);           /* Init pointer to CUR descr */
      2          STAT_DESC(AVE_STAT) = ADDR(QUAL$L_AVE);           /* Init pointer to AVE descr */
      2          STAT_DESC(MIN_STAT) = ADDR(QUAL$L_MIN);           /* Init pointer to MIN descr */
      2          STAT_DESC(MAX_STAT) = ADDR(QUAL$L_MAX);           /* Init pointer to MAX descr */
      2
  !   2          /*
  !   2          /*      Initialize PROCD_TABLE descriptor pointers.
  !   2          /*/
      2
      2          PROCD_DESC(REG_PROC) = NULL();                    /* Indicate no qualifier for "regular" PROCESSES display */
      2          PROCD_DESC(TOPC_PROC) = ADDR(QUAL$L_TOPC);        /* Init pointer to TOPC descr */
      2          PROCD_DESC(TOPD_PROC) = ADDR(QUAL$L_TOPD);        /* Init pointer to TOPD descr */
      2          PROCD_DESC(TOPB_PROC) = ADDR(QUAL$L_TOPB);        /* Init pointer to TOPB descr */
      2          PROCD_DESC(TOPF_PROC) = ADDR(QUAL$L_TOPF);        /* Init pointer to TOPF descr */
      2
      2          MRB_DEF.DIS = ADDR(DEF$L_DISP);                   /* Init default value of ptr to display file descr */
      2          ALLOCATE MCA SET(MCAPTR);                         /* Obtain Monitor Communication Area */
      2          ALLOCATE SYS_INFO SET(SPTR);                      /* Obtain System Information Area */
      2
  !   2          /*
  !   2          /*      Allocate "temp", "current" and "active" MRBs. Another MRB (MRB_DEF),
  !   2          /*      the "default" MRB, is allocated in the data section above. Each MRB fully describes
  !   2          /*      a MONITOR request. The usage of the MRBs is analogous to that of SYSGEN PAR files.
  !   2          /*      To wit, the "default" MRB never changes; it is moved to the "current" MRB whenever
  !   2          /*      an INITIALIZE subcommand is issued; an implicit INITIALIZE occurs when the MONITOR
  !   2          /*      image is invoked. The "current" MRB may be changed repeatedly with SET commands without
  !   2          /*      actually executing a request; When a MONITOR request is executed (with a MONITOR
  !   2          /*      subcommand, or a MONITOR DCL command), the "current" MRB is moved to the "active" MRB,
  !   2          /*      the "active" MRB is updated in accordance with the MONITOR request, and the request
  !   2          /*      is executed using the "active" MRB. The "temp" MRB is used to accumulate information
  !   2          /*      resulting from a SET command. If the command contains no errors, "temp" is swapped
  !   2          /*      with "current"; otherwise, "temp" is discarded. This is done to avoid defining a partial
  !   2          /*      request when the SET command contains an error.
  !   2          /*/
      2
      2          ALLOCATE MRB SET(TEMP_MRBPTR);                    /* Allocate the "temp" MRB */
      2          ALLOCATE MRB SET(CURR_MRBPTR);                    /* Allocate the "current" MRB */
      2          ALLOCATE MRB SET(ACT_MRBPTR);                     /* Allocate the "active" MRB */
      2          DEF_MRBPTR = ADDR(MRB_DEF);                       /* Init pointer to "default" MRB */
      2
      2          CALL = INIT_CMD();                                /* Do an INITIALIZE command ... i.e., move default */
  :   2                                                            /* ... MRB to current MRB */
      2
      2          RETURN(CALL);                                     /* Return with status from INIT_CMD */
      2          END MONITOR_INIT;
      1
```

```
 1       NEXT_COMMAND: Procedure Returns(Fixed Binary(31));      /* Routine to read the next subcommand & execute it */
 2
 2 :     /***
 2 :     /*
 2 :     /* FUNCTIONAL DESCRIPTION:
 2 :     /*
 2 :     /*      NEXT_COMMAND
 2 :     /*
 2 :     /*      This routine is called by the main routine to read the next subcommand and execute it.
 2 :     /*      It returns a status code indicating the disposition of the command.
 2 :     /*
 2 :     /* INPUTS:
 2 :     /*
 2 :     /*      None
 2 :     /*
 2 :     /* IMPLICIT INPUTS:
 2 :     /*
 2 :     /*      PROMPT -- a bit indicating whether or not to prompt for a command.
 2 :     /*                It has always been set to YES before entry to NEXT_COMMAND.
 2 :     /*
 2 :     /* OUTPUTS:
 2 :     /*
 2 :     /*      None
 2 :     /*
 2 :     /* IMPLICIT OUTPUTS:
 2 :     /*
 2 :     /*      The next command is executed.
 2 :     /*
 2 :     /*      PROMPT  -- a bit indicating whether or not to prompt for another
 2 :     /*      command. It is always set to YES before entry to NEXT_COMMAND, and
 2 :     /*      can be set to NO by NEXT_COMMAND in any of the following situations:
 2 :     /*
 2 :     /*              1) The user issues the EXIT subcommand.
 2 :     /*              2) The user strikes CTRL/Z in response to the MONITOR> prompt.
 2 :     /*              3) The user strikes CTRL/Z while a MONITOR subcommand is running.
 2 :     /*
 2 :     /*      PUTMSGVEC -- a 20-longword vector which is loaded with error message
 2 :     /*                   information by NEXT_COMMAND whenever an error status is
 2 :     /*                   returned to the caller of NEXT_COMMAND.
 2 :     /*
 2 :     /* ROUTINE VALUE:
 2 :     /*
 2 :     /*      A status code indicating the disposition of the command. If an
 2 :     /*      error status, then the PUTMSGVEC error message vector will have
 2 :     /*      been set up.
 2 :     /*
 2 :     /* SIDE EFFECTS:
 2 :     /*
 2 :     /*      None
 2 :     /*
 2 :     /**/
 2
```

```
/*
/*      +----------------------------------------------------------------------+
/*      |                                                                      |
/*      |                     COMPILE TIME DEFINITIONS                         |
/*      |                                                                      |
/*      +----------------------------------------------------------------------+
/*/

%REPLACE        MAX_EXEC_LINE           BY 256;                 /* Max execute file line size */

/*
/*      +----------------------------------------------------------------------+
/*      |                                                                      |
/*      |                          LOCAL STORAGE                               |
/*      |                                                                      |
/*      +----------------------------------------------------------------------+
/*/

Declare
  CLI$DCL_PARSE EXTERNAL ENTRY(ANY, ANY)                        /* Rtn to do DCL-like parsing of cmd line */
                RETURNS(FIXED BINARY(31)),
  CLI$DISPATCH  EXTERNAL ENTRY                                  /* Rtn to dispatch to latest parsed cmd */
                RETURNS(FIXED BINARY(31)),
  LIB$GET_INPUT EXTERNAL ENTRY(ANY, ANY, FIXED BINARY(15))      /* Rtn to read a line from SYS$INPUT */
                OPTIONS(VARIABLE)
                RETURNS(FIXED BINARY(31)),
  STR$POSITION  EXTERNAL ENTRY(ANY, ANY, ANY)                   /* Rtn to locate 1st occurance of a substring */
                OPTIONS(VARIABLE)
                RETURNS(FIXED BINARY(31)),
  STR$REPLACE   EXTERNAL ENTRY(ANY, ANY, ANY, ANY, ANY)         /* Rtn to replace a substring with another substring
                RETURNS(FIXED BINARY(31));

Declare
MNR$_ERREXEREP  FIXED BINARY(31) GLOBALREF VALUE,               /* Error message code */
MNR$_ERRPROMPT  FIXED BINARY(31) GLOBALREF VALUE;               /* Error message code */

/*
/*      String descriptor for user prompt string
/*/

Declare

1 PROMPT_STR GLOBALREF,                                         /* User prompt string descriptor */
  2 L           FIXED BINARY(31),
  2 A           POINTER;

Declare
1 DYN_STRING    GLOBALREF,                                      /* Dynamic string descriptor */
  2 L           FIXED BINARY(15),                               /* Length */
  2 TC          CHAR(2),                                        /* Type and Class */
  2 A           POINTER,                                        /* Address */
DYN_STRING_S    CHAR(DYN_STRING.L) BASED(DYN_STRING.A);         /* String */

Declare
1 CMD_LINE,                                                     /* Static command line descriptor (for subcommands)
```

```
      2          2 L              FIXED BINARY(31),                                       /* Length */
      2          2 A              POINTER,                                                /* Address */
      2
      2        1 REP_LINE,                                                                /* Static command line descriptor (for subcommands)
      2          2 L              FIXED BINARY(31),                                       /* Length */
      2          2 A              POINTER,                                                /* Address */
      2
      2        CMD_LINE_S         CHAR(MAX_EXEC_LINE) STATIC;                             /* Command buffer to replace '@' with 'EXECUTE ' */
      2
      2
      2        Declare
      2          MONSUB           CHAR(1) GLOBALREF;                                      /* Command language definition tables ... */
   :  2                                                                                   /* Note -- we simply need a global reference to the
   :  2                                                                                   /* tables. Their length is unknown and irrelevant */
      2        Declare
      2          AT_SIGN_S        CHAR(1) STATIC INIT('@'),                               /* "@" used to search command line */
      2          EXECUTE_S        CHAR(8) STATIC INIT('EXECUTE '),                        /* "EXECUTE " used to replace the "@" in the command
      2          AT_SIGN_POS      FIXED BINARY(31);                                       /* Position of "@" in the command line */
      2
      2
      2        FIRST_MON_CMD = NO;                                                        /* First MONITOR cmd executes before NEXT_COMMAND rt
      2        DYN_STRING.L = 0;                                                          /* Init cmd line length to enter loop */
      2
      2        DO WHILE (DYN_STRING.L = 0);                                               /* Loop while user enters null lines */
      3        CALL = LIB$GET_INPUT(DYN_STRING,PROMPT_STR,);                              /* Read the next subcommand */
      3        IF STATUS = NOT_SUCCESSFUL                                                 /* If LIB$GET_INPUT call failed, */
      3           THEN DO;
      4                PROMPT = NO;                                                       /* Indicate no more prompting */
      4                IF CALL = RMS$_EOF                                                 /* If end-of-input, */
      4                   THEN RETURN(NORMAL);                                            /*    then return with normal status */
      4                   ELSE DO;
      5                        CALL MON_ERR(MNR$_ERRPROMPT,CALL);                         /* Otherwise, log the error ... */
      5                        RETURN(MNR$_ERRPROMPT);                                    /* and return with status */
      5                        END;
      4                END;
      3        END;
      2
      2        CMD_LINE.L = DYN_STRING.L;                                                 /* Copy the length of the command line */
      2        CMD_LINE.A = ADDR(CMD_LINE_S);                                             /* Get the address of the new working buffer */
      2        CMD_LINE_S = DYN_STRING_S;                                                 /* Copy the command line into the buffer */
      2        AT_SIGN_POS = STR$POSITION(CMD_LINE, DESCRIPTOR(AT_SIGN_S));               /* Locate a "@" in the command line */
      2        IF AT_SIGN_POS > 0                                                         /* Was there one? */
      2           THEN DO;                                                                /* Yes, prepare to replace the "@" with 'EXECUTE ' *
      3               REP_LINE.L = CMD_LINE.L + 7;                                        /* Add space for 'EXECUTE ' in  replacement desc */
      3               REP_LINE.A = CMD_LINE.A;                                            /* Get address of replacement string */
      3               CALL = STR$REPLACE (REP_LINE, CMD_LINE, AT_SIGN_POS,                /* Replace the "@" with 'EXECITE ' */
      3                                   AT_SIGN_POS, DESCRIPTOR(EXECUTE_S));/*  (need REP_LINE to prevent trunc warning) */
      3               IF STATUS = NOT_SUCCESSFUL                                          /* If STR$REPLACE call failed, */
      3                   THEN DO;
      4                        CALL MON_ERR(MNR$_ERREXEREP,CALL);                         /* Log the error ... */
      4                        RETURN(MNR$_ERREXEREP);                                    /* and return with status */
      4                        END;
      3                   ELSE CMD_LINE.L = REP_LINE.L;                                   /* STR$REPLACE succeeded, update length of descr */
      3               END;
      2
```

```
  2          CURR_ERRCODE = MNR$_ERRPARSE;                                        /* Set MONITOR code in case parsing error signaled *
  2          CALL = CLI$DCL_PARSE(CMD_LINE,MONSUB);                               /* Parse the subcommand */
  2          CURR_ERRCODE = 0;                                                    /* Reset to default MONITOR code */
  2          IF STATUS = NOT_SUCCESSFUL                                           /* If parse failed, */
  2              THEN DO;
  3                   IF CALL = CLI$_NOCOMD                                       /* If "no command on line" */
  3                       THEN RETURN(NORMAL);                                    /*    then quietly ignore it */
  3                       ELSE DO;                                                /* Otherwise, */
  4                            CALL MON_ERR(MNR$_ERRPARSE,CALL);                  /*    log the error ... */
  4                            RETURN(MNR$_ERRPARSE);                             /*    ... and return with status */
  4                            END;
  3                   END;
  2
  2          CALL = CLI$DISPATCH();                                               /* Execute the parsed command */
: 2                                                                               /* Note -- command subroutines return status */
: 2                                                                               /*          values and log their own errors by */
: 2                                                                               /*          calling MON_ERR */
  2
  2          CURR_ERRCODE = 0;                                                    /* Reset to default MONITOR code in case subcommand
  2          RETURN(CALL);                                                        /* Return to caller with cmd subroutine's status */
  2
  2          END NEXT_COMMAND;
  1
  1
```

```
   1       NEXT_EXECUTE_COMMAND: Procedure Returns(Fixed Binary(31));       /* Routine to read the next subcommand & execute it
   2
 : 2       /***
 : 2       /*
 : 2       /* FUNCTIONAL DESCRIPTION:
 : 2       /*
 : 2       /*      NEXT_EXECUTE_COMMAND
 : 2       /*
 : 2       /*      This routine is called by the main routine to read the next subcommand from
 : 2       /*      the execute command file and execute it.  It returns a status code indicating
 : 2       /*      the disposition of the command.
 : 2       /*
 : 2       /* INPUTS:
 : 2       /*
 : 2       /*      None
 : 2       /*
 : 2       /* IMPLICIT INPUTS:
 : 2       /*
 : 2       /*      COMMAND_FILE - file reference used to read the execute command file.
 : 2       /*
 : 2       /*      EXECUTE - a bit indicating whether or not commands are read from an execute
 : 2       /*                command file.  It has been set to YES before entry to NEXT_EXECUTE_COMMAND.
 : 2       /*
 : 2       /* OUTPUTS:
 : 2       /*
 : 2       /*      None
 : 2       /*
 : 2       /* IMPLICIT OUTPUTS:
 : 2       /*
 : 2       /*      The next command is executed.
 : 2       /*
 : 2       /*      EXECUTE - a bit indicating whether or not to read from the execute command file
 : 2       /*      for another command.  It is always set to YES before entry to NEXT_EXECUTE_COMMAND,
 : 2       /*      and can be set to NO by NEXT_EXECUTE_COMMAND in any of the following situations:
 : 2       /*
 : 2       /*              1) Execute command file EOF is encountered.
 : 2       /*              2) The user issues the EXIT subcommand.
 : 2       /*              3) The user strikes CTRL/Z in response to the MONITOR> prompt.
 : 2       /*              4) The user strikes CTRL/Z while a MONITOR subcommand is running.
 : 2       /*              5) The file cannot be opened properly.
 : 2       /*              6) An error occurs when trying to read the file.
 : 2       /*
 : 2       /*      PUTMSGVEC -- a 20-longword vector which is loaded with error message
 : 2       /*                   information by NEXT_EXECUTE_COMMAND whenever an error status is
 : 2       /*                   returned to the caller of NEXT_EXECUTE_COMMAND.
 : 2       /*
 : 2       /* ROUTINE VALUE:
 : 2       /*
 : 2       /*      A status code indicating the disposition of the command. If an error status,
 : 2       /*      then the PUTMSGVEC error message vector will have been set up.
 : 2       /*
 : 2       /*      MNR$_ERRPARSE    - error parsing command from the execute command file.
 : 2       /*      MNR$_ERREXEREA   - error reading execute command file.
 : 2       /*      SS$_NORMAL       - Success.
 : 2       /*
 : 2       /* SIDE EFFECTS:
```

```
 1674 |  2        /*
 1675 |  2        /*        None.
 1676 |  2        /*
 1677 |  2        /*/
 1678    2
 !  2        /*
 !  2        /*      +------------------------------------------------------------------------+
 !  2        /*      |                                                                        |
 !  2        /*      |                      COMPILE TIME DEFINITIONS                          |
 !  2        /*      |                                                                        |
 !  2        /*      +------------------------------------------------------------------------+
 !  2        /*/
    2
    2        %REPLACE        MAX_EXEC_LINE           BY 256;                 /* Max execute file line size */
    2
 !  2        /*
 !  2        /*      +------------------------------------------------------------------------+
 !  2        /*      |                                                                        |
 !  2        /*      |                      EXTERNAL STORAGE DEFINITIONS                      |
 !  2        /*      |                                                                        |
 !  2        /*      +------------------------------------------------------------------------+
 !  2        /*/
    2
    2        Declare
    2            CLI$DCL_PARSE       EXTERNAL ENTRY(ANY, ANY)                /* Rtn to do DCL-like parsing of cmd line */
    2                                RETURNS(FIXED BINARY(31)),
    2            CLI$DISPATCH        EXTERNAL ENTRY                          /* Rtn to dispatch to latest parsed cmd */
    2                                RETURNS(FIXED BINARY(31)),
    2            CLI$GET_VALUE       ENTRY(ANY, ANY)                         /* CLE routine to get qualifier values */
    2                                RETURNS(FIXED BINARY(31)),
    2            STR$POSITION        EXTERNAL ENTRY(ANY, ANY, ANY)           /* Rtn to locate 1st occurance of a substring */
    2                                OPTIONS(VARIABLE)
    2                                RETURNS(FIXED BINARY(31)),
    2            STR$REPLACE         EXTERNAL ENTRY(ANY, ANY, ANY, ANY, ANY) /* Rtn to replace a substring with another substring
    2                                RETURNS(FIXED BINARY(31));
    2
    2
    2        Declare
    2            COMMAND_FILE        FILE               GLOBALREF,           /* Execute Command File */
    2            MONSUB              CHAR(1)            GLOBALREF,           /* Command language definition tables ... */
 !  2                                                                        /* Note -- we simply need a global reference to the
 !  2                                                                        /* tables. Their length is unknown and irrelevant */
    2            NORMAL              FIXED BINARY(31) GLOBALREF,             /* MONITOR normal return status */
    2            EXECUTE             BIT(1) ALIGNED     GLOBALREF;           /* YES => read another command from the execute file
    2
    2        Declare
    2            MNR$_ERREXEREP      FIXED BINARY(31) GLOBALREF VALUE,       /* Error message code */
    2            MNR$_ERREXEREA      FIXED BINARY(31) GLOBALREF VALUE,       /* Error message code */
    2            MNR$_ERRPARSE       FIXED BINARY(31) GLOBALREF VALUE;       /* Error message code */
    2
    2
 !  2        /*
 !  2        /*      +------------------------------------------------------------------------+
 !  2        /*      |                                                                        |
 !  2        /*      |                      GLOBAL STORAGE DEFINITIONS                        |
 !  2        /*      |                                                                        |
 !  2        /*      +------------------------------------------------------------------------+
 !  2        /*/
    2
    2        Declare
    2            1 SUB_COMMAND       GLOBALDEF,
     2               2 L                 FIXED BINARY(31),
     2               2 A                 POINTER,
     2             COMMAND_RECORD      CHAR(MAX_EXEC_LINE) STATIC VARYING GLOBALDEF;   /* Buffer for a subcommand */
     2
     2
   | 2        /*
   | 2        /*       +---------------------------------------------------------------------------+
   | 2        /*       |                                                                           |
   | 2        /*       |                            LOCAL STORAGE                                  |
   | 2        /*       |                                                                           |
   | 2        /*       +---------------------------------------------------------------------------+
   | 2        /*/
     2
     2
     2        Declare
     2             1 REP_LINE,                                                   /* Static command line descriptor (for subcommands)
     2               2 L                 FIXED BINARY(31),                       /* Length */
     2               2 A                 POINTER;                                /* Address */
     2
     2        Declare
     2             AT_SIGN_S    CHAR(1) STATIC INIT('@'),                        /* "@" used to search command line */
     2             EXECUTE_S    CHAR(8) STATIC INIT('EXECUTE '),                 /* "EXECUTE " used to replace "@" */
     2             AT_SIGN_POS FIXED BINARY(31),                                 /* Position of "@" in command line */
     2             TEMP_COMMAND_PTR    FIXED BINARY(31)                          /* Alias for SUB_COMMAND.A computation */
     2                                 BASED(ADDR(SUB_COMMAND.A));
     2
     2
     2        ON ENDFILE (COMMAND_FILE) GOTO COMMAND_EOF;                        /* Set up the EOF condition */
     2
     2
     2        CURR_ERRCODE = MNR$_ERREXEREA;                                     /* Set MONITOR code in case read error is signaled *
     2        READ FILE (COMMAND_FILE) INTO (COMMAND_RECORD);                    /* Read the next subcommand */
     2        CURR_ERRCODE = 0;                                                  /* Reset the error code, assume the condition was re
     2
     2        SUB_COMMAND.L = LENGTH(COMMAND_RECORD);                            /* Set the length of the SUB_COMMAND descriptor */
     2        SUB_COMMAND.A = ADDR(COMMAND_RECORD);                              /* Set the address of the SUB_COMMAND descriptor */
     2        TEMP_COMMAND_PTR = TEMP_COMMAND_PTR + 2;                           /* Advance ptr beyond length word */
     2        AT_SIGN_POS = STR$POSITION(SUB_COMMAND, DESCRIPTOR(AT_SIGN_S));    /* Locate "@" in command line */
     2        IF AT_SIGN_POS > 0                                                 /* Was there a "@" ? */
     2             THEN DO;                                                      /* Yes, prepare to replace "@" with "EXECUTE " */
     3                  REP_LINE.L = SUB_COMMAND.L + 7;                          /* Add space for "EXECUTE " in replacement string */
     3                  REP_LINE.A = SUB_COMMAND.A;                              /* Get address of subcommand buffer */
     3                  CALL = STR$REPLACE (REP_LINE, SUB_COMMAND, AT_SIGN_POS,  /* Replace "@" with "EXECUTE " */
     3                                 AT_SIGN_POS, DESCRIPTOR(EXECUTE_S));      /*  (REP_LINE prevents truncation warning) */
     3                  IF STATUS = NOT_SUCCESSFUL                               /* If STR$REPLACE call failed, */
     3                       THEN DO;
     4                            CALL MON_ERR(MNR$_ERREXEREP,CALL);             /* Log the error ... */
     4                            RETURN(MNR$_ERREXEREP);                        /* and return with status */
     4                            END;
     3                       ELSE SUB_COMMAND.L = REP_LINE.L;                    /* STR$REPACE succeded, upade command desc */
     3                  END;
     2
```

```
       2        CURR_ERRCODE = MNR$_ERRPARSE;                                     /* Set MONITOR code in case parsing error signaled *
       2        CALL = CLI$DCL_PARSE(SUB_COMMAND,MONSUB);                         /* Parse the subcommand */
       2        CURR_ERRCODE = 0;                                                 /* Reset to default MONITOR code */
       2        IF STATUS = NOT_SUCCESSFUL                                        /* If parse failed, */
       2            THEN DO;
       3                IF CALL = CLI$_NOCOMD                                     /* If "no command on line" */
       3                    THEN RETURN(NORMAL);                                  /*    then quietly ignore it */
       3                    ELSE DO;                                              /* Otherwise, */
       4                        CALL MON_ERR(MNR$_ERRPARSE,CALL);                 /*    log the error ... */
       4                        RETURN(MNR$_ERRPARSE);                            /*    ... and return with status */
       4                        END;
       3                END;
       2
       2        CALL = CLI$DISPATCH();                                            /* Execute the parsed command */
   :   2                                                                          /* Note -- command subroutines return status */
   :   2                                                                          /*         values and log their own errors by */
   :   2                                                                          /*         calling MON_ERR */
       2        CURR_ERRCODE = 0;                                                 /* Reset to default MONITOR code in case subcommand
       2        RETURN(CALL);                                                     /* Return to caller with cmd subroutine's status */
       2
       2
       2        COMMAND_EOF:
       2            CLOSE FILE (COMMAND_FILE);                                    /* Close the file after EOF condition raised */
       2            EXECUTE = NO;                                                 /* Indicate no more from the execute file */
       2            CURR_ERRCODE = 0;                                             /* Reset to default MONITOR code in case subcommand
       2            RETURN(NORMAL);                                               /* Return to caller with cmd subroutine's status */
       2
       2        END NEXT_EXECUTE_COMMAND;
       1
       1        END MONMAIN;
```

```
        GET_QUALIFIERS: Procedure Returns(Fixed Binary(31));
    1
    1   /***
    1   /*
    1   /* FUNCTIONAL DESCRIPTION:
    1   /*
    1   /*       GET_QUALIFIERS
    1   /*
    1   /*       Communicate with CLE to get qualifier settings and
    1   /*       their values. Record all such info in the MRB (Monitor
    1   /*       Request Block) pointed to by the current value of MRBPTR
    1   /*       (it may be the "current" or the "active" MRB).
    1   /*
    1   /* INPUTS:
    1   /*
    1   /*       TBS
    1   /*
    1   /* IMPLICIT INPUTS:
    1   /*
    1   /*       TBS
    1   /*
    1   /* OUTPUTS:
    1   /*
    1   /*       TBS
    1   /*
    1   /* IMPLICIT OUTPUTS:
    1   /*
    1   /*       TBS
    1   /*
    1   /* ROUTINE VALUE:
    1   /*
    1   /*       TBS
    1   /*
    1   /* SIDE EFFECTS:
    1   /*
    1   /*       TBS
    1   /*
    1   /*/
    1
```

```
   |  1        /*
   |  1        /*      +------------------------------------------------------------------------------+
   |  1        /*      |                                                                              |
   |  1        /*      |                              INCLUDE  FILES                                  |
   |  1        /*      |                                                                              |
   |  1        /*      +------------------------------------------------------------------------------+
   |  1        /**/
      1
      1        %INCLUDE        MONDEF;                         /* Monitor utility structure definitions */
      1
   |  1        /*
   |  1        /*      +------------------------------------------------------------------------------+
   |  1        /*      |                                                                              |
   |  1        /*      |                SYSTEM SERVICE MACRO DEFINITIONS                              |
   |  1        /*      |                                                                              |
   |  1        /*      +------------------------------------------------------------------------------+
   |  1        /**/
      1
      1        %INCLUDE        SYS$GETTIM;                     /* $GETTIM system service */
   |  1        /*
   |  1        /*      +------------------------------------------------------------------------------+
   |  1        /*      |                                                                              |
   |  1        /*      |                   EXTERNAL STORAGE  DEFINITIONS                              |
   |  1        /*      |                                                                              |
   |  1        /*      +------------------------------------------------------------------------------+
   |  1        /**/
      1
      1        Declare
      1        CLI$_PRESENT    FIXED BINARY(31) GLOBALREF VALUE,               /* CLI$PRESENT return status code for "explicitly pr
      1        CLI$_NEGATED    FIXED BINARY(31) GLOBALREF VALUE,               /* CLI$PRESENT return status code for "explicitly ne
      1        NORMAL          FIXED BINARY(31) GLOBALREF,                     /* MONITOR normal return status */
      1        MRBPTR          POINTER          GLOBALREF,                     /* Pointer to MRB (Monitor Request Block) */
      1        M               POINTER          DEFINED(MRBPTR),               /* Short-hand synonym for MRBPTR */
      1        DEF_MRBPTR      POINTER          GLOBALREF,                     /* Pointer to "default" MRB (Monitor Request Block)
      1        MCAPTR          POINTER          GLOBALREF,                     /* Pointer to MCA (Monitor Communication Area) */
      1        MC              POINTER          DEFINED(MCAPTR),               /* Short-hand synonym for MCAPTR */
      1        QUALPTR         POINTER          GLOBALREF,                     /* Pointer to Qualifier Descriptors Block */
      1        DEFPTR          POINTER          GLOBALREF,                     /* Pointer to Qualifier Default Value Descriptors Bl
      1        CURR_ERRCODE    FIXED BINARY(31) GLOBALREF;                     /* MONITOR error status code currently expected */
      1
      1        Declare
      1        INP_PTR_VOL     POINTER GLOBALREF,                              /* Pointer to volatile /INPUT file-spec */
      1        DISP_PTR_VOL    POINTER GLOBALREF,                              /* Pointer to volatile /DISPLAY file-spec */
      1        REC_PTR_VOL     POINTER GLOBALREF,                              /* Pointer to volatile /RECORD file-spec */
      1        SUMM_PTR_VOL    POINTER GLOBALREF,                              /* Pointer to volatile /SUMMARY file-spec */
      1        COMM_PTR_VOL    POINTER GLOBALREF;                              /* Pointer to volatile /COMMENT string */
      1
      1        Declare
      1        INP_PTR_SWAP    BIT(1) ALIGNED GLOBALREF,                       /* YES => swap INP_PTR_VOL and INP_PTR_PERM */
      1        DISP_PTR_SWAP   BIT(1) ALIGNED GLOBALREF,                       /* YES => swap DISP_PTR_VOL and DISP_PTR_PERM */
      1        REC_PTR_SWAP    BIT(1) ALIGNED GLOBALREF,                       /* YES => swap REC_PTR_VOL and REC_PTR_PERM */
      1        SUMM_PTR_SWAP   BIT(1) ALIGNED GLOBALREF,                       /* YES => swap SUMM_PTR_VOL and SUMM_PTR_PERM */
      1        COMM_PTR_SWAP   BIT(1) ALIGNED GLOBALREF;                       /* YES => swap COMM_PTR_VOL and COMM_PTR_PERM */
      1
```

```
Declare
1 DYN_STRING      GLOBALREF,                                   /* Dynamic string descriptor */
  2 L             FIXED BINARY(15),                            /* Length */
  2 TC            CHAR(2),                                     /* Type and Class */
  2 A             POINTER,                                     /* Address */

DYN_STRING_S      CHAR(DYN_STRING.L) BASED(DYN_STRING.A);      /* String */

/*
/*      +----------------------------------------------------------------+
/*      |                                                                |
/*      |                 EXTERNAL ROUTINE DEFINITIONS                   |
/*      |                                                                |
/*      +----------------------------------------------------------------+
/*/

Declare
CLI$GET_VALUE     ENTRY(ANY, ANY, FIXED BINARY(15))            /* CLE routine to get qualifier values */
                  OPTIONS(VARIABLE) RETURNS(BIT(1)),
CLI$PRESENT       EXTERNAL ENTRY(ANY) RETURNS(FIXED BINARY(31)), /* CLE routine to determine presence of qualifiers *
LIB$CVT_TIME      EXTERNAL ENTRY(ANY, BIT(64) ALIGNED)         /* RTL routine to convert to qword time val */
                  RETURNS(BIT(1)),
STR$UPCASE        EXTERNAL ENTRY(ANY,ANY) RETURNS(BIT(1)),     /* RTL routine to upcase a string */
MON_ERR           ENTRY (ANY VALUE, ANY, ANY) OPTIONS(VARIABLE); /* MONITOR MACRO-32 routine to log synchronous error

/*
/*      +----------------------------------------------------------------+
/*      |                                                                |
/*      |                     MESSAGE DEFINITIONS                        |
/*      |                                                                |
/*      +----------------------------------------------------------------+
/*/

Declare
MNR$_INVBTIMSP  FIXED BINARY(31) GLOBALREF VALUE,      /* Error message code */
MNR$_INVETIMSP  FIXED BINARY(31) GLOBALREF VALUE,      /* Error message code */
MNR$_INVINTSP   FIXED BINARY(31) GLOBALREF VALUE,      /* Error message code */
MNR$_INVFLUSHSP FIXED BINARY(31) GLOBALREF VALUE,      /* Error message code */
MNR$_INVVIEWSP  FIXED BINARY(31) GLOBALREF VALUE,      /* Error message code */
MNR$_INVINPFIL  FIXED BINARY(31) GLOBALREF VALUE;      /* Error message code */
/*
/*      +----------------------------------------------------------------+
/*      |                                                                |
/*      |                  GLOBAL STORAGE DEFINITIONS                    |
/*      |                                                                |
/*      +----------------------------------------------------------------+
/*/

Declare
QUAL_SPECIFIED  BIT(1) ALIGNED GLOBALDEF INIT('0'B);   /* YES => at least 1 qualifier explicitly spec'd */

/*
/*      +----------------------------------------------------------------+
/*      |                                                                |
/*      |                    COMPILE-TIME CONSTANTS                      |
/*      |                                                                |
   : 1       /*      !------------------------------------------------------------------------------+
   : 1       /*/
     1
     1       %REPLACE        SECONDS_TOK_SIZE        BY 7;           /* Size of token for seconds */
     1       %REPLACE        TIME_TOK_SIZE           BY 40;          /* Size of token for time specs */
     1       %REPLACE        FILE_SPEC_SIZE          BY 128;         /* Max file spec size */
     1
```

```
/*
/*      +----------------------------------------------------------+
/*      |                                                          |
/*      |                       OWN STORAGE                        |
/*      |                                                          |
/*      +----------------------------------------------------------+
/*/

/*
/*      +----------------------------------------------------------+
/*      |                                                          |
/*      |                      LOCAL STORAGE                       |
/*      |                                                          |
/*      +----------------------------------------------------------+
/*/

Declare
TEMP                    FIXED BINARY(31),                           /* Temporary "scratch" area */
CALL                    FIXED BINARY(31),                           /* Holds function value (return status) of called ro
STATUS                  BIT(1) BASED(ADDR(CALL));                   /* Low-order status bit for called routines */

Declare
FILE_SPEC_PTR           POINTER,                                    /* Pointer to structure consisting of file-spec ...
                                                                    /* ... string descriptor followed by string itself *
PARSED_SPEC_PTR         POINTER,                                    /* Pointer to structure consisting of file-spec ...
                                                                    /* ... string descriptor followed by string itself *
COMM_STR_PTR            POINTER;                                    /* Pointer to structure consisting of comment ... */
                                                                    /* ... string descriptor followed by string itself *

Declare
1 FILE_SPEC     BASED(FILE_SPEC_PTR),                               /* File-spec string descriptor and string */
    2 L         FIXED BINARY(31),                                   /* File-spec string length */
    2 A         POINTER,                                            /* File-spec string address */
    2 S         CHAR(FILE_SPEC_SIZE),                               /* File-spec string */


1 PARSED_SPEC   BASED(PARSED_SPEC_PTR),                             /* Parsed File-spec dynamic string descriptor and st
    2 L         FIXED BINARY(15),                                   /* Length */
    2 T         FIXED BINARY(7),                                    /* Type */
    2 C         FIXED BINARY(7),                                    /* Class */
    2 A         POINTER,                                            /* Address */


1 DYN_SPEC      GLOBALDEF,                                          /* Dynamic File-spec string descriptor and string */
    2 L         FIXED BINARY(15),                                   /* Length */
    2 T         FIXED BINARY(7),                                    /* Type */
    2 C         FIXED BINARY(7),                                    /* Class */
    2 A         POINTER;                                            /* Address */


Declare
1 COMM_STR      BASED(COMM_STR_PTR),                                /* Comment descriptor and string */
    2 L         FIXED BINARY(31),                                   /* Comment string length */
    2 A         POINTER,                                            /* Comment string address */
    2 S         CHAR(MNR_HDR$K_MAXCOMLEN);                          /* Comment string */
```

```
     1      Declare
     1        1 Q_VAL,                                             /* String descriptor for qualifier value */
     1          2 L                  FIXED BINARY(31),              /* Length */
     1          2 A                  POINTER,                       /* Address */
     1
     1        Q_VAL_TIME             CHAR(TIME_TOK_SIZE),           /* Qualifier value string for time values */
     1        Q_VAL_SECS             CHAR(SECONDS_TOK_SIZE+1),      /* Qualifier value string for seconds values */
     1        TS_LEN                 FIXED BINARY(15);              /* Actual length of time spec */
     1
     1
```

```
 1
 1      QUAL_SPECIFIED = NO;                                                       /* No qualifiers explicitly specified yet */
 1
 1      CALL = SYS$GETTIM(MC->MCA$Q_CURR_TIME);                                    /* Get current time from system */
 1
 1      Q_VAL.L = LENGTH(Q_VAL_TIME);                                              /* Set length field of descriptor */
 1      Q_VAL.A = ADDR(Q_VAL_TIME);                                                /* Set address field of descriptor */
 1
 1      CALL = CLI$PRESENT(QUAL$L_BEG);                                            /* Get BEGINNING qualifier presence indicato
 1      IF CALL = CLI$_PRESENT                                                     /* If explicitly present, */
 1        THEN
 1          DO;
 2          QUAL_SPECIFIED = YES;                                                  /* Indicate qualifier explicitly specified *
 2          IF CLI$GET_VALUE(QUAL$L_BEG,Q_VAL,TS_LEN)                              /* Get "BEGINNING" string and check status *
 2          THEN DO;                                                               /* Value was specified */
 3               Q_VAL.L = TS_LEN;                                                 /* Pick up actual length */
 3               IF Q_VAL.L > TIME_TOK_SIZE                                        /* Check for valid size for time spec */
 3                   THEN DO;
 4                        CALL MON_ERR(MNR$_INVBTIMSP);                            /* Log possible error */
 4                        RETURN(MNR$_INVBTIMSP);                                  /* ... and return with status */
 4                        END;
 3               IF ^ STR$UPCASE(Q_VAL,Q_VAL)                                      /* Upcase and check status */
 3                   THEN DO;
 4                        CALL MON_ERR(MNR$_INVBTIMSP);                            /* Log possible error */
 4                        RETURN(MNR$_INVBTIMSP);                                  /* ... and return with status */
 4                        END;
 3               IF ^ LIB$CVT_TIME(Q_VAL,M->MRB$Q_BEGINNING)                       /* Cvt to system time */
 3                   THEN DO;
 4                        CALL MON_ERR(MNR$_INVBTIMSP);                            /* Log possible error */
 4                        RETURN(MNR$_INVBTIMSP);                                  /* ... and return with status */
 4                        END;
 3               END;
 2
 2          ELSE                                                                   /* Value was defaulted */
 2               M->MRB$Q_BEGINNING = DEF_MRBPTR->MRB$Q_BEGINNING;                 /* Store the default value */
 2          END;
 1
 1      Q_VAL.L = LENGTH(Q_VAL_TIME);                                              /* Set length field of descriptor */
 1
 1      CALL = CLI$PRESENT(QUAL$L_END);                                            /* Get ENDING qualifier presence indicator *
 1      IF CALL = CLI$_PRESENT                                                     /* If explicitly present, */
 1        THEN
 1          DO;
 2          QUAL_SPECIFIED = YES;                                                  /* Indicate qualifier explicitly specified *
 2          IF CLI$GET_VALUE(QUAL$L_END,Q_VAL,TS_LEN)                              /* Get "ENDING" string and check status */
 2          THEN DO;                                                               /* Value was specified */
 3               Q_VAL.L = TS_LEN;                                                 /* Pick up actual length */
 3               IF Q_VAL.L > TIME_TOK_SIZE                                        /* Check for valid size for time spec */
 3                   THEN DO;
 4                        CALL MON_ERR(MNR$_INVETIMSP);                            /* Log possible error */
 4                        RETURN(MNR$_INVETIMSP);                                  /* ... and return with status */
 4                        END;
 3               IF ^ STR$UPCASE(Q_VAL,Q_VAL)                                      /* Upcase and check status */
 3                   THEN DO;
 4                        CALL MON_ERR(MNR$_INVETIMSP);                            /* Log possible error */
 4                        RETURN(MNR$_INVETIMSP);                                  /* ... and return with status */
```

```
      4                         END;
      3             IF ^ LIB$CVT_TIME(Q_VAL,M->MRB$Q_ENDING)                                /* Cvt to system time */
      3                 THEN DO;
      4                         CALL MON_ERR(MNR$_INVETIMSP);                                /* Log possible error */
      4                         RETURN(MNR$_INVETIMSP);                                      /* ... and return with status */
      4                         END;
      3             END;
      2
      2         ELSE                                                                         /* Value was defaulted */
      2             M->MRB$Q_ENDING = DEF_MRBPTR->MRB$Q_ENDING;                              /* Store the default value */
      1         END;
```

```
 1      Q_VAL.L = LENGTH(Q_VAL_SECS);                                      /* Set length field of descriptor */
 1      Q_VAL.A = ADDR(Q_VAL_SECS);                                        /* Set address field of descriptor */
 1
 1      CALL = CLI$PRESENT(QUAL$L_INT);                                    /* Get INTERVAL qualifier presence indicator
 1      IF CALL = CLI$_PRESENT                                             /* If explicitly present, */
 1        THEN
 1          DO;
 2          QUAL_SPECIFIED = YES;                                          /* Indicate qualifier explicitly specified *
 2          IF CLI$GET_VALUE(QUAL$L_INT,Q_VAL)                             /* Get "INTERVAL" string and check status */
 2          THEN DO;                                                       /* Value was specified */
 3                  Q_VAL.L = INDEX(Q_VAL_SECS,' ') - 1;                   /* Eliminate trailing blanks */
 3                  IF Q_VAL.L <= 0                                        /* Check for valid size for "seconds" */
 3                  THEN DO;
 4                       CALL MON_ERR(MNR$_INVINTSP);                      /* Log possible error */
 4                       RETURN(MNR$_INVINTSP);                            /* ... and return with status */
 4                       END;
 3                  CURR_ERRCODE = MNR$_INVINTSP;                          /* Set MONITOR code in case conversion error
 3                  M->MRB$L_INTERVAL = BIN(SUBSTR(Q_VAL_SECS,1,Q_VAL.L),31);  /* Convert seconds to binary */
 3                  CURR_ERRCODE = 0;                                      /* Reset to default MONITOR code */
 3                  IF M->MRB$L_INTERVAL <= 0                              /* Check for valid value */
 3                  THEN DO;
 4                       CALL MON_ERR(MNR$_INVINTSP);                      /* Log possible error */
 4                       RETURN(MNR$_INVINTSP);                            /* ... and return with status */
 4                       END;
 3              END;
 2
 2          ELSE                                                           /* Value was defaulted */
 2              M->MRB$L_INTERVAL = DEF_MRBPTR->MRB$L_INTERVAL;            /* Store the default value */
 2          END;
 1
 1      Q_VAL.L = LENGTH(Q_VAL_SECS);                                      /* Set length field of descriptor */
 1
 1      CALL = CLI$PRESENT(QUAL$L_FLUSH);                                  /* Get FLUSH qualifier presence indicator */
 1      IF CALL = CLI$_PRESENT                                             /* If explicitly present, */
 1        THEN
 1          DO;
 2          QUAL_SPECIFIED = YES;                                          /* Indicate qualifier explicitly specified *
 2          IF CLI$GET_VALUE(QUAL$L_FLUSH,Q_VAL)                           /* Get "FLUSH" string and check status */
 2          THEN DO;                                                       /* Value was specified */
 3                  Q_VAL.L = INDEX(Q_VAL_SECS,' ') - 1;                   /* Eliminate trailing blanks */
 3                  IF Q_VAL.L <= 0                                        /* Check for valid size for "seconds" */
 3                  THEN DO;
 4                       CALL MON_ERR(MNR$_INVFLUSHSP);                           /* Log possible error */
 4                       RETURN(MNR$_INVFLUSHSP);                          /* ... and return with status */
 4                       END;
 3                  CURR_ERRCODE = MNR$_INVFLUSHSP;                        /* Set MONITOR code in case conversion error
 3                  M->MRB$L_FLUSH = BIN(SUBSTR(Q_VAL_SECS,1,Q_VAL.L),31);     /* Convert seconds to binary */
 3                  CURR_ERRCODE = 0;                                      /* Reset to default MONITOR code */
 3                  IF M->MRB$L_FLUSH <= 0                                 /* Check for valid value */
 3                  THEN DO;
 4                       CALL MON_ERR(MNR$_INVFLUSHSP);                           /* Log possible error */
 4                       RETURN(MNR$_INVFLUSHSP);                          /* ... and return with status */
 4                       END;
 3              END;
 2
 2          ELSE                                                           /* Value was defaulted */
```

```
    2                   M->MRB$L_FLUSH = DEF_MRBPTR->MRB$L_FLUSH;                   /* Store the default value */
    2           END;
    1
    1       Q_VAL.L = LENGTH(Q_VAL_SECS);                                           /* Set length field of descriptor */
    1
    1       CALL = CLI$PRESENT(QUAL$L_VIEW);                                        /* Get VIEWING_TIME qualifier presence indic
    1       IF CALL = CLI$_PRESENT                                                  /* If explicitly present, */
    1         THEN
    1           DO;
    2           QUAL_SPECIFIED = YES;                                               /* Indicate qualifier explicitly specified *
    2           IF CLI$GET_VALUE(QUAL$L_VIEW,Q_VAL)                                 /* Get "VIEWING_TIME" string and check statu
    2           THEN DO;                                                            /* Value was specified */
    3                   Q_VAL.L = INDEX(Q_VAL_SECS,' ') - 1;                        /* Eliminate trailing blanks */
    3                   IF Q_VAL.L <= 0                                             /* Check for valid size for "seconds" */
    3                   THEN DO;
    4                        CALL MON_ERR(MNR$_INVVIEWSP);                          /* Log possible error */
    4                        RETURN(MNR$_INVVIEWSP);                                /* ... and return with status */
    4                        END;
    3                   CURR_ERRCODE = MNR$_INVVIEWSP;                              /* Set MONITOR code in case conversion error
    3                   M->MRB$L_VIEWING_TIME = BIN(SUBSTR(Q_VAL_SECS,1,Q_VAL.L),31); /* Convert seconds to binary */
    3                   CURR_ERRCODE = 0;                                           /* Reset to default MONITOR code */
    3                   IF M->MRB$L_VIEWING_TIME <= 0                               /* Check for valid value */
    3                   THEN DO;
    4                        CALL MON_ERR(MNR$_INVVIEWSP);                          /* Log possible error */
    4                        RETURN(MNR$_INVVIEWSP);                                /* ... and return with status */
    4                        END;
    3                END;
    2
    2           ELSE                                                                /* Value was defaulted */
    2                M->MRB$L_VIEWING_TIME = DEF_MRBPTR->MRB$L_VIEWING_TIME;        /* Store the default value */
    2           END;
    1
    1
```

```
   1        CALL = CLI$PRESENT(QUAL$L_BY_NODE);                               /* Get BY_NODE qualifier presence indicator
   1        IF CALL = CLI$_PRESENT                                            /* If explicitly present, */
   1          THEN
   1             DO;
   2             QUAL_SPECIFIED = YES;                                        /* Indicate qualifier explicitly specified *
   2             M->MRB$V_BY_NODE = YES;                                      /* Turn on flag in MRB */
   2             END;
   1
   1        IF CALL = CLI$_NEGATED                                            /* If explicitly negated, */
   1             THEN DO;
   2                   QUAL_SPECIFIED = YES;                                  /* Indicate qualifier explicitly specified *
   2                   M->MRB$V_BY_NODE = NO;                                 /* Turn off flag in MRB */
   2                   END;
   1
   1        CALL = CLI$PRESENT(QUAL$L_INP);                                   /* Get INPUT qualifier presence indicator */
   1        IF CALL = CLI$_PRESENT                                            /* If explicitly present, */
   1             THEN DO;
   2                   QUAL_SPECIFIED = YES;                                  /* Indicate qualifier explicitly specified *
   2                   M->MRB$V_MFSUM = NO;                                   /* Set to NO for now (may change in BUILD_IF
   2                   CALL = BUILD_IFB_TABLE();                              /* Build the table of Input File Blocks (IFB
   2                   IF ^STATUS
   2                   THEN RETURN(CALL);                                     /* return with error status, MON_ERR was alr
   2                   END;
   1        IF CALL = CLI$_NEGATED                                            /* If explicitly negated, */
   1             THEN DO;
   2                   QUAL_SPECIFIED = YES;                                  /* Indicate qualifier explicitly specified *
   2                   M->MRB$V_MFSUM = NO;                                   /* Turn off Multi-File Summary indicator */
   2                   M->MRB$A_INPUT = NULL();                               /* Indicate no input file */
   2                   END;
   1
```

```
1       CALL = CLI$PRESENT(QUAL$L_DISP);                                          /* Get DISPLAY qualifier presence indicator
1       IF CALL = CLI$_PRESENT                                                    /* If explicitly present, */
1           THEN DO;
2                QUAL_SPECIFIED = YES;                                            /* Indicate qualifier explicitly specified *
2                IF DISP_PTR_VOL = NULL()                                         /* If no volatile file spec string area, */
2                    THEN DO;
3                         ALLOCATE FILE_SPEC;                                     /*   then get one */
3                         DISP_PTR_VOL = FILE_SPEC_PTR;                           /*   set up a ptr to it */
3                         FILE_SPEC.L = LENGTH(FILE_SPEC.S);                      /*   set length ... */
3                         FILE_SPEC.A = ADDR(FILE_SPEC.S);                        /*   and address in descriptor */
3                         END;
2
2                    ELSE DO;
3                         FILE_SPEC_PTR = DISP_PTR_VOL;                           /* Otherwise, simply point to existing one *
3                         FILE_SPEC.L = FILE_SPEC_SIZE;                           /* ... and re-init its length */
3                         END;
2
2                IF CLI$GET_VALUE(QUAL$L_DISP,FILE_SPEC)                          /* Qualifier value specified ? */
2                    THEN DO;
3                         DISP_PTR_SWAP = YES;                                    /* Yes -- ind to SET_CMD a ptr swap is neces
3                         TEMP = INDEX(FILE_SPEC.S,' ') - 1;                      /* Find trailing blanks in value (string) */
3                         IF TEMP >= 0 THEN FILE_SPEC.L = TEMP;                   /* If found one, set new length */
3                         M->MRB$A_DISPLAY = FILE_SPEC_PTR;                       /* Store away pointer to value descr */
3                         M->MRB$V_DISP_TO_FILE = YES;                            /* Indicate filespec specified */
3                         END;
2
2                    ELSE DO;                                                     /* No -- qualifier value defaulted */
3                         M->MRB$A_DISPLAY = ADDR(DEF$L_DISP);                    /* Store a default value descr */
3                         M->MRB$V_DISP_TO_FILE = NO;                             /* ... and default indicator */
3                         END;
2
2                END;
1
1       IF CALL = CLI$_NEGATED                                                    /* If explicitly negated, */
1           THEN DO;
2                QUAL_SPECIFIED = YES;                                            /* Indicate qualifier explicitly specified *
2                M->MRB$A_DISPLAY = NULL();                                       /* Indicate no display output */
2                M->MRB$V_DISP_TO_FILE = NO;                                      /* ...... */
2                END;
1
1       CALL = CLI$PRESENT(QUAL$L_REC);                                           /* Get RECORD qualifier presence indicator *
1       IF CALL = CLI$_PRESENT                                                    /* If explicitly present, */
1           THEN DO;
2                QUAL_SPECIFIED = YES;                                            /* Indicate qualifier explicitly specified *
2                IF REC_PTR_VOL = NULL()                                          /* If no volatile file spec string area, */
2                    THEN DO;
3                         ALLOCATE FILE_SPEC;                                     /*   then get one */
3                         REC_PTR_VOL = FILE_SPEC_PTR;                            /*   set up a ptr to it */
3                         FILE_SPEC.L = LENGTH(FILE_SPEC.S);                      /*   set length ... */
3                         FILE_SPEC.A = ADDR(FILE_SPEC.S);                        /*   and address in descriptor */
3                         END;
2
2                    ELSE FILE_SPEC_PTR = REC_PTR_VOL;                            /* Otherwise, simply point to existing one *
2
2                IF CLI$GET_VALUE(QUAL$L_REC,FILE_SPEC)                           /* Qualifier value specified ? */
2                    THEN DO;
```

```
       3                    REC_PTR_SWAP = YES;                              /* Yes -- ind to SET_CMD a ptr swap is neces
       3                    M->MRB$A_RECORD = FILE_SPEC_PTR;                 /* Store away pointer to string descr */
       3                    END;
       2
       2              ELSE M->MRB$A_RECORD = ADDR(DEF$L_REC);                /* No -- store a default value */
       2
       2          END;
       1
       1  IF CALL = CLI$_NEGATED                                             /* If explicitly negated, */
       1      THEN DO;
       2          QUAL_SPECIFIED = YES;                                      /* Indicate qualifier explicitly specified *
       2          M->MRB$A_RECORD = NULL();                                  /* Indicate no record output */
       2          END;
       1
       1  CALL = CLI$PRESENT(QUAL$L_SUMM);                                   /* Get SUMMARY qualifier presence indicator
       1  IF CALL = CLI$_PRESENT                                             /* If explicitly present, */
       1      THEN DO;
       2          QUAL_SPECIFIED = YES;                                      /* Indicate qualifier explicitly specified *
       2          IF SUMM_PTR_VOL = NULL()                                   /* If no volatile file spec string area, */
       2              THEN DO;
       3                    ALLOCATE FILE_SPEC;                              /*   then get one */
       3                    SUMM_PTR_VOL = FILE_SPEC_PTR;                    /*   set up a ptr to it */
       3                    FILE_SPEC.L = LENGTH(FILE_SPEC.S);               /*   set length ... */
       3                    FILE_SPEC.A = ADDR(FILE_SPEC.S);                 /*   and address in descriptor */
       3                    END;
       2
       2              ELSE FILE_SPEC_PTR = SUMM_PTR_VOL;                     /* Otherwise, simply point to existing one *
       2
       2          IF CLI$GET_VALUE(QUAL$L_SUMM,FILE_SPEC)                    /* Qualifier value specified ? */
       2              THEN DO;
       3                    SUMM_PTR_SWAP = YES;                             /* Yes -- ind to SET_CMD a ptr swap is neces
       3                    M->MRB$A_SUMMARY = FILE_SPEC_PTR;                /* Store away pointer to string descr */
       3                    END;
       2
       2              ELSE M->MRB$A_SUMMARY = ADDR(DEF$L_SUMM);              /* No -- store a default value */
       2
       2          END;
       1
       1  IF CALL = CLI$_NEGATED                                             /* If explicitly negated, */
       1      THEN DO;
       2          QUAL_SPECIFIED = YES;                                      /* Indicate qualifier explicitly specified *
       2          M->MRB$A_SUMMARY = NULL();                                 /* Indicate no summary output */
       2          END;
       1
       1  CALL = CLI$PRESENT(QUAL$L_COMM);                                   /* Get COMMENT qualifier presence indicator
       1  IF CALL = CLI$_PRESENT                                             /* If explicitly present, */
       1      THEN DO;
       2          QUAL_SPECIFIED = YES;                                      /* Indicate qualifier explicitly specified *
       2          IF COMM_PTR_VOL = NULL()                                   /* If no volatile comment string area, */
       2              THEN DO;
       3                    ALLOCATE COMM_STR;                               /*   then get one ... */
       3                    COMM_PTR_VOL = COMM_STR_PTR;                     /*   and set up a ptr to it */
       3                    END;
       2
       2              ELSE COMM_STR_PTR = COMM_PTR_VOL;                      /* Otherwise, simply point to existing one *
       2
       2          IF CLI$GET_VALUE(QUAL$L_COMM,DYN_STRING)                   /* Qualifier value specified ? */
```

```
2                   THEN DO;
3                        COMM_STR.S = DYN_STRING_S;                      /* Yes -- move string out of dyn area */
3                        COMM_STR.L = DYN_STRING.L;                      /* ... and set up its length */
3                        COMM_STR.A = ADDR(COMM_STR.S);                  /* ... and address */
3                        COMM_PTR_SWAP = YES;                            /* Ind to SET_CMD a ptr swap is necessary */
3                        M->MRB$A_COMMENT = COMM_STR_PTR;                /* Store away pointer to string descr */
3                        END;
2
2                   ELSE M->MRB$A_COMMENT = NULL();                      /* No -- store a default value */
2
2               END;
1
1     IF CALL = CLI$_NEGATED                                             /* If explicitly negated, */
1         THEN DO;
2               QUAL_SPECIFIED = YES;                                    /* Indicate qualifier explicitly specified *
2               M->MRB$A_COMMENT = NULL();                               /* Indicate no comment string */
2               END;
1
1     RETURN(NORMAL);                                                    /* Return with status */
1
```

```
   1        BUILD_IFB_TABLE: Procedure Returns(Fixed Binary(31));
   2
   2        /*++
   2        /*
   2        /* FUNCTIONAL DESCRIPTION:
   2        /*
   2        /*      BUILD_IFB_TABLE
   2        /*
   2        /*      This routine builds the IFB (Input File Block) TABLE.
   2        /*      In addition, it sets up MRB$A_INPUT to point to the IFB TABLE, and
   2        /*      sets up MRB$B_INP_FILES to be the number of input files described by
   2        /*      the table.
   2        /*
   2        /* INPUTS:
   2        /*
   2        /*      None
   2        /*
   2        /* IMPLICIT INPUTS:
   2        /*
   2        /*      IFB_TABLE, MRB
   2        /*
   2        /* OUTPUTS:
   2        /*
   2        /*      None
   2        /*
   2        /* IMPLICIT OUTPUTS:
   2        /*
   2        /*      IFB_TABLE built, MRB$A_INPUT and MRB$B_INP_FILES estblished.
   2        /*
   2        /* ROUTINE VALUE:
   2        /*
   2        /*      Normal, or bad status from LIB$FIND_FILE
   2        /*
   2        /* SIDE EFFECTS:
   2        /*
   2        /*      None
   2        /*
   2        /*/
   2
```

```
 !   2        /*
 !   2        /*        +-----------------------------------------------------------------------+
 !   2        /*        |                                                                       |
 !   2        /*        |                     GLOBAL STORAGE DEFINITIONS                        |
 !   2        /*        |                                                                       |
 !   2        /*        +-----------------------------------------------------------------------+
 !   2        /**/
     2
     2        Declare
     2        MAX_INP_FILES          FIXED BINARY(31) GLOBALDEF VALUE INIT(125); /* Max no. of input files for multi-file summary
     2
     2
 !   2        /*
 !   2        /*        +-----------------------------------------------------------------------+
 !   2        /*        |                                                                       |
 !   2        /*        |                         EXTERNAL REFERENCES                           |
 !   2        /*        |                                                                       |
 !   2        /*        +-----------------------------------------------------------------------+
 !   2        /**/
     2
     2        %INCLUDE        MONDEF;                                          /* Monitor utility structure definitions */
     2
     2        Declare
     2        MON_ERR         ENTRY (ANY VALUE, ANY, ANY) OPTIONS(VARIABLE),
 !   2                                                                /* Routine to log synchronous errors */
     2        DSC$K_DTYPE_T   FIXED BINARY(15) GLOBALREF VALUE,        /* String  descr. type */
     2        DSC$K_CLASS_D   FIXED BINARY(15) GLOBALREF VALUE,        /* Dynamic descr. class */
     2        SS$_NORMAL      FIXED BINARY(31) GLOBALREF VALUE,        /* System normal return status */
     2        RMS$_EOF        FIXED BINARY(31) GLOBALREF VALUE,        /* RMS end-of-file return status */
     2        RMS$_NMF        FIXED BINARY(31) GLOBALREF VALUE,        /* RMS no-more-files message for wildcard parsing */
     2        RMS$_FNF        FIXED BINARY(31) GLOBALREF VALUE,        /* RMS file-not-found message */
     2        MNR$_OPENIN     FIXED BINARY(31) GLOBALREF VALUE,        /* Error message code */
     2        MNR$_TOOMNYFILES FIXED BINARY(31) GLOBALREF VALUE,       /* Error message code */
     2        LIB$FIND_FILE   EXTERNAL ENTRY(ANY,ANY,ANY,ANY,ANY,ANY,ANY)     /* RTL routine to parse a wildcard spec*/
     2                        OPTIONS(VARIABLE) RETURNS(FIXED BINARY(31)),
     2        LIB$FIND_FILE_END EXTERNAL ENTRY(ANY),                   /* RTL routine to end wildcard spec parse*/
     2        LIB$SCOPY_DXDX  EXTERNAL ENTRY(ANY,ANY),                 /* RTL string copy routine */
     2        IFB_TAB_VOL     POINTER GLOBALREF,                       /* Pointer to volatile IFB_TABLE */
     2        IFB_TAB_SWAP    BIT(1) ALIGNED GLOBALREF;                /* YES => swap IFB_TAB_VOL and IFB_TAB_PERM */
     2
 !   2        /*
 !   2        /*        +-----------------------------------------------------------------------+
 !   2        /*        |                                                                       |
 !   2        /*        |                            LOCAL STORAGE                              |
 !   2        /*        |                                                                       |
 !   2        /*        +-----------------------------------------------------------------------+
 !   2        /**/
     2
     2        Declare
     2        USER_FLAGS      FIXED BINARY(31) INIT(2),                        /* Flag indicating "stickiness desired" */
     2        CONTEXT         FIXED BINARY(31),
     2        CALL            FIXED BINARY(31),                                /* Holds function value (return status) of called ro
     2        STATUS          BIT(1) BASED(ADDR(CALL)),                        /* Low-order status bit for called routines */
     2        FIND_FILE_CALL  FIXED BINARY(31),                                /* Holds function value (return status) of LIB$FIND_
     2        FIND_FILE_STAT  BIT(1) BASED(ADDR(FIND_FILE_CALL)),              /* Low-order status bit for LIB$FIND_FILE */
     2        SPEC_LEN        FIXED BINARY(15),                                /* Input filespec length */
     2          I               FIXED BINARY(7),                                /* Loop control */
     2          VALUE_FOUND     BIT(1) ALIGNED,                                 /* YES => a value for the /INPUT qualifier found */
     2          IFBPTR          POINTER,                                        /* Pointer to Input File Block (IFB) */
     2          IFB_TAB_PTR     POINTER;                                        /* Pointer to IFB_TABLE */
     2
     2          Declare
     2          1 IFB_TABLE     BASED(IFB_TAB_PTR),                             /* Input File Block (IFB) Table */
     2            2 AR_IFB      (1:MAX_INP_FILES+1)  CHAR(IFB$K_SIZE);          /* A single IFB */
     2
     2          Declare
     2          REC_DEF_S       CHAR(DEF$L_REC) BASED(DEF$A_REC);               /* String for default recording file spec */
     2
     2
     2    IF IFB_TAB_VOL = NULL()                                          /* If no volatile IFB table, */
     2       THEN DO;
     3          ALLOCATE IFB_TABLE;                                         /*   then get one */
     3          IFB_TAB_VOL = IFB_TAB_PTR;                                  /*   set up a ptr to it */
     3          DO I = 1 TO MAX_INP_FILES + 1;                              /*   clear entire */
     4          UNSPEC(AN_IFB(I)) = '0'B;                                   /*   array to */
     4          END;                                                        /*   zeroes */
     3          END;
     2
     2       ELSE IFB_TAB_PTR = IFB_TAB_VOL;                                /* Otherwise, simply point to existing one */
     2
   ! 2    /*
   ! 2    /*      NOTE -- at this point, the volatile pointer (IFB_TAB_VOL) and the base
   ! 2    /*              pointer (IFB_TAB_PTR) both point to the IFB_TABLE in use.
   ! 2    /*/
     2    M->MRB$B_INP_FILES = 0;                                           /* Start off loop with no input files */
     2    VALUE_FOUND = YES;                                                /* ... and assume a qualifier value (file spec) found */
     2    CALL = SS$_NORMAL;                                                /* Init main loop status */
     2    I = 1;                                                            /* Init input file counter */
     2    CONTEXT = 0;                                                      /* init CONTEXT */
   ! 2    /*
   ! 2    /*      NOTE -- What follows are two loops - the outer loop does CLI$GET_VALUE calls, and the inner loop does
   ! 2    /*              LIB$FIND_FILE calls. The CLI$GET_VALUE loop is controlled by STATUS, which is the low bit of CALL, a
   ! 2    /*              also by VALUE_FOUND. CALL will always be SS$_NORMAL unless the LIB$FIND_FILE loop runs into trouble.
   ! 2    /*              CALL is what is ultimately returned by this procedure.
   ! 2    /*/
     2    DO WHILE(STATUS & VALUE_FOUND);                                   /* Begin CLI$GET_VALUE loop */
     3       FIND_FILE_CALL = SS$_NORMAL;                                   /* init FIND_FILE status */
     3       DYN_SPEC.T = DSC$K_DTYPE_T;                                    /* Init  str. descr. type */
     3       DYN_SPEC.C = DSC$K_CLASS_D;                                    /* dynamic class */
     3       IF ^CLI$GET_VALUE(QUAL$L_INP,DYN_SPEC)                         /* File spec specified ? */
     3       THEN DO;                                                       /* no filespec specified with qualifier */
     4          IF M->MRB$B_INP_FILES = 0                                   /* If first time around */
     4          THEN CALL LIB$SCOPY_DXDX(DEF$L_REC,DYN_SPEC);               /* use default filespec */
     4          ELSE VALUE_FOUND = NO;                                      /* else indicate no more filespecs to skip LIB$FIND_FILE loo
     4          END;
     3       DO WHILE(STATUS & FIND_FILE_STAT & VALUE_FOUND);               /* Begin LIB$FIND_FILE loop */
     4          IFBPTR = ADDR(AN_IFB(I));                                   /* Address an IFB */
     4          IF IFB$A_INPUT = NULL()                                     /* If not pointing to a file-spec yet, */
     4          THEN DO;
     5             ALLOCATE PARSED_SPEC;                                    /*   allocate space for result */
     5             IFB$A_INPUT = PARSED_SPEC_PTR;                           /*   set up a ptr to it */
     5             PARSED_SPEC.T = DSC$K_DTYPE_T;                           /*   str. descr. type */
     5             PARSED_SPEC.C = DSC$K_CLASS_D;                           /*   dynamic class */
     5             PARSED_SPEC.A = NULL();                                  /*   make sure length, */
     5             PARSED_SPEC.L = 0;                                       /*   and address are 0 */
     5             END;
     4          ELSE PARSED_SPEC_PTR = IFB$A_INPUT;                         /* Otherwise, simply point to existing one */
     4          FIND_FILE_CALL=LIB$FIND_FILE(DYN_SPEC,PARSED_SPEC,CONTEXT,DEF$L_REC,,,USER_FLAGS); /* Get the next full file s
/
     4             IF FIND_FILE_STAT                                        /* Did we get another valid filespec? */
     4                THEN DO;
     5                   M->MRB$B_INP_FILES = M->MRB$B_INP_FILES + 1;       /* Yes -- count it */
     5                   IF I > MAX_INP_FILES                               /* If we have exceeded the max. allowed # of input f
     5                      THEN DO;
```

```
6                       CALL MON_ERR(MNR$_TOOMNYFILES);                 /* then log the error */
6                       CALL = MNR$_TOOMNYFILES;                        /* set bad status to end both inner and outer loops
6                       END;
5                       ELSE I = I + 1;                                 /*...else increment file count and continue the loop
5                  END;
4           END;                                                        /* end of LIB$FIND_FILE loop */
3           IF (^FIND_FILE_STAT & FIND_FILE_CALL ^= RMS$_NMF)            /* "no-more-files" is the only valid error from LIB$
3                THEN DO;                                               /* ...any other status means trouble (such as file n
4                     CALL MON_ERR(MNR$_OPENIN,FIND_FILE_CALL,PARSED_SPEC);/* log error */
4                     CALL = MNR$_OPENIN;                               /* set bad status to end both the inner and outer lo
4                     END;
3       END;                                                            /* End of CLI$GET_VALUE loop */
2           CALL LIB$FIND_FILE_END(CONTEXT);                            /* wipe out context of prev LIB$FIND_FILE calls */
2       IFB_TAB_SWAP = YES;                                             /* Ind to SET_CMD a ptr swap is necessary */
2       M->MRB$A_INPUT = IFB_TAB_PTR;                                   /* Store ptr to IFB table */
2       IF M->MRB$B_INP_FILES > 1  THEN M->MRB$V_MFSUM = YES;            /* If more than 1 file spec, indicate multi-file sum
2
2       RETURN (CALL);                                                  /* Return to caller */
2
2       END BUILD_IFB_TABLE;
1
1       END GET_QUALIFIERS;
```

```
             GET_CLASSES: Procedure (AT_LEAST_ONE_CLASS) Returns(Fixed Binary(31));
        1
   :    1    /***
   :    1    /*
   :    1    /* FUNCTIONAL DESCRIPTION:
   :    1    /*
   :    1    /*      GET_CLASSES
   :    1    /*
   :    1    /*      Communicate with CLE to get class_name keywords; then use keywords as
   :    1    /*      input to LOOKUP_KEY and get back class numbers. Ultimate output is
   :    1    /*      MRB$O_CLASSBITS, a bit string for which each bit number corresponds
   :    1    /*      to class number. If a lookup error is found, bad status is returned
   :    1    /*      to the issuer.
   :    1    /*
   :    1    /* INPUTS:
   :    1    /*
   :    1    /*      TBS
   :    1    /*
   :    1    /* IMPLICIT INPUTS:
   :    1    /*
   :    1    /*      TBS
   :    1    /*
   :    1    /* OUTPUTS:
   :    1    /*
   :    1    /*      TBS
   :    1    /*
   :    1    /* IMPLICIT OUTPUTS:
   :    1    /*
   :    1    /*      TBS
   :    1    /*
   :    1    /* ROUTINE VALUE:
   :    1    /*
   :    1    /*      TBS
   :    1    /*
   :    1    /* SIDE EFFECTS:
   :    1    /*
   :    1    /*      TBS
   :    1    /*
   :    1    /**/
        1
```

```
/*
/*      !------------------------------------------------------------------------------------+
/*      !                                                                                    !
/*      !                              LOCAL STORAGE                                         !
/*      !                                                                                    !
/*      !------------------------------------------------------------------------------------+
/*/

Declare
AT_LEAST_ONE_CLASS        BIT(1) ALIGNED,                          /* YES => user requested at least one class */
CALL                      FIXED BINARY(31),                        /* Holds function value (return status) of called ro
STATUS                    BIT(1) BASED(ADDR(CALL)),                /* Low-order status bit for called routines */
CLASS_KEY                 FIXED BINARY(31) STATIC,                 /* Class keyword number */
                                                                   /* Note -- must be STATIC to get error msg */
TEMP                      FIXED BINARY(31);                        /* Scratch "register" */


%REPLACE        NOT_SUCCESSFUL          BY '0'B;                   /* Failing severity code */
%REPLACE        CLASS_TOK_SIZE          BY 20;                     /* Size of token for class-name */
%REPLACE        OR_OP                   BY '0111'B;                /* OR Boolean operation */

/*
/*      !------------------------------------------------------------------------------------+
/*      !                                                                                    !
/*      !                           EXTERNAL REFERENCES                                      !
/*      !                                                                                    !
/*      !------------------------------------------------------------------------------------+
/*/

%INCLUDE        MONDEF;                                            /* Monitor utility structure definitions */

Declare
CLI$GET_VALUE   ENTRY(ANY, ANY)                                    /* CLE routine to get qualifier values */
                RETURNS(BIT(1)),
LIB$LOOKUP_KEY  EXTERNAL ENTRY(ANY, ANY, FIXED BINARY(31))         /* RTL rtn to look up class-names in a table */
                RETURNS(BIT(1)),
MON_ERR         ENTRY (ANY VALUE, ANY, ANY) OPTIONS(VARIABLE);     /* MONITOR MACRO-32 routine to log synchronous error

Declare
I               FIXED BINARY(7),                                   /* Loop control */
MAX_CLASS_NO    FIXED BINARY(31) GLOBALREF VALUE,                  /* Maximum defined class number */
ALL_CLSNO       FIXED BINARY(31) GLOBALREF VALUE,                  /* ALL class Pseudo class number */
ALL_CLASS_FOUND BIT(1),                                            /* Flag to indicate ALL class found on command line
MNR$_INVCLSNM   FIXED BINARY(31) GLOBALREF VALUE,                  /* Error message code */
NORMAL          FIXED BINARY(31) GLOBALREF,                        /* MONITOR normal return status */
CDBPTR          POINTER          GLOBALREF,                        /* Pointer to CDB (Class Descriptor Block) */
C               POINTER          DEFINED(CDBPTR),                  /* Short-hand synonym for CDBPTR */
MRBPTR          POINTER          GLOBALREF,                        /* Pointer to MRB (Monitor Request Block) */
M               POINTER          DEFINED(MRBPTR),                  /* Short-hand synonym for MRBPTR */
QUALPTR         POINTER          GLOBALREF;                        /* Pointer to Qualifier Descriptors Block */

Declare
1 CDBHEAD GLOBALREF,                                               /* Table of CDB's */
  2 CDBLOCK (0:127) CHAR(CDB$K_SIZE);

Declare
```

```
     1      CLASSTABLE      CHAR(1)         GLOBALREF;                       /* Table of class names & numbers (dummy) */
  !  1                                                                       /* Note -- CLASSTABLE is declared here simply */
  !  1                                                                       /*    so its address can be referenced below */
     1
     1      Declare
     1      REQUEST_CLASS_MASK      BIT(MAX_CLASS_NO+1),            /* Requested classes have bits on */
     1      REQUEST_CLASS_VEC       (0:MAX_CLASS_NO) BIT(1) DEFINED(REQUEST_CLASS_MASK);
  !  1                                                              /* Alias for REQUEST_CLASS_MASK, but bit-addressable */
     1
     1      Declare
     1      01 CLASS_VAL STATIC,                                     /* String descr for value of CLASS_NAME qualifier */
  !  1                                                              /* Note -- must be STATIC to get INVCLSNM msg */
     1          02 L                FIXED BINARY(31),               /* Length */
     1          02 A                POINTER,                        /* Address */
     1          02 S                CHAR(CLASS_TOK_SIZE+1);         /* String */
     1
     1
     1      CLASS_VAL.L = LENGTH(CLASS_VAL.S);                       /* Init length longword of descr */
     1      CLASS_VAL.A = ADDR(CLASS_VAL.S);                         /* Init address longword of descr */
     1
     1      REQUEST_CLASS_MASK = '0'B;                               /* Turn off all class bits initially */
     1      ALL_CLASS_FOUND = NO;                                    /* Assume we won't find ALL pseudo-class on this command lin
     1      DO WHILE(CLI$GET_VALUE(QUAL$L_CLASS,CLASS_VAL));         /* Loop once for each requested class */
     2          CLASS_VAL.L = INDEX(CLASS_VAL.S,' ') - 1;            /* Now strip off trailing blanks */
     2          IF CLASS_VAL.L < 0 THEN CLASS_VAL.L = CLASS_TOK_SIZE; /* If too long, replace with max token size */
     2
     2          IF ^ LIB$LOOKUP_KEY(CLASS_VAL,CLASSTABLE,CLASS_KEY) /* Get class keyword number */
     2              THEN DO;
     3                   CALL MON_ERR(MNR$_INVCLSNM,,CLASS_VAL);     /* Log error if bad class name */
     3                   RETURN(MNR$_INVCLSNM);                      /* ... and return with status */
     3                   END;
     2          CLASS_VAL.L = CLASS_TOK_SIZE + 1;                    /* Restore string len for next loop */
     2          IF CLASS_KEY = ALL_CLSNO                             /* If all classes  */
     2              THEN DO;
     3                   ALL_CLASS_FOUND = YES;                      /* inidicate we found ALL on this command line */
     3                   M->MRB$V_ALL_CLASS = YES;                   /* indicate its ALL class*/
     3                      DO I = 0 TO MAX_CLASS_NO;                /* Loop once for each possible class */
     4                      REQUEST_CLASS_VEC(I) = YES;              /* Turn on bit for this class */
     4                      CDBPTR = ADDR(CDBLOCK(I));               /* Get CDB addressability */
     4                      IF C->CDB$V_DISABLE THEN REQUEST_CLASS_VEC(I) = NO; /* If this class disabled, then ignore it */
     4                      END;
     3                   END;
     2              ELSE REQUEST_CLASS_VEC(CLASS_KEY) = YES;         /* Turn on bit for this class */
     2          CDBPTR = ADDR(CDBLOCK(CLASS_KEY));                   /* Get CDB addressability */
     2          IF C->CDB$V_DISABLE                                  /* If this class disabled, */
     2              THEN REQUEST_CLASS_VEC(CLASS_KEY) = NO;          /*    then ignore it */
     2              ELSE DO;                                         /* Otherwise, */
     3                   CALL = GET_CLASS_QUALS(CLASS_KEY);          /* Process class qualifiers for this class */
     3                   IF STATUS = NOT_SUCCESSFUL THEN RETURN (CALL); /* ... and check status */
     3                   END;
     2      END;
     1
     1      IF INDEX(REQUEST_CLASS_MASK,'1'B) ^= 0                   /* If any classes specified, */
     1          THEN DO;
     2               AT_LEAST_ONE_CLASS = YES;                       /* indicate got at least one */
     2               IF ALL_CLASS_FOUND = NO                         /* if we didn't find the ALL pseudo-class */
     2               THEN  M->MRB$V_ALL_CLASS = NO;                  /* ...make sure the ALL class flag is clear (this logic is n
: 2                                                                /* for defaulting to work correctly in interactive mode) */
  2               M->MRB$O_CLASSBITS = REQUEST_CLASS_MASK;         /* ... move selected bits */
  2               END;
  1         ELSE DO;                                               /* No classes specified, use MRB$O_CLASSBITS */
  2               IF INDEX(M->MRB$O_CLASSBITS,'1'B) = 0                    /* If no classes specified there, */
  2                   THEN AT_LEAST_ONE_CLASS = NO;                        /*   then indicate so */
  2                   ELSE AT_LEAST_ONE_CLASS = YES;                       /*   else indicate got one or more */
  2               END;
  1
  1     RETURN(NORMAL);                                            /* Return to caller */
  1
  1
```

```
     1       GET_CLASS_QUALS: Procedure (CLSNO) Returns (fixed binary(31));
     2
     2       Declare
     2       CLI$PRESENT      ENTRY(ANY) RETURNS(BIT(1)),                          /* CLE routine to determine presence of quals */
     2       CLI$GET_VALUE    ENTRY(ANY, ANY, FIXED BINARY(15))                    /* CLE routine to get qualifier values */
     2                        RETURNS(BIT(1));
     2
     2       Declare
     2       MNR$_QUALINV     FIXED BINARY(31) GLOBALREF VALUE,                    /* Error message code */
     2       MNR$_SQUALERR    FIXED BINARY(31) GLOBALREF VALUE,                    /* Error message code */
     2       MNR$_PDQUALERR   FIXED BINARY(31) GLOBALREF VALUE,                    /* Error message code */
     2       MNR$_INVITEMNM   FIXED BINARY(31) GLOBALREF VALUE;                    /* Error message code */
     2
     2       Declare
     2       MODES_CLSNO      FIXED BINARY(31) GLOBALREF VALUE;                    /* MODES class number */
     2
     2       Declare
     2       TEMP_CDBPTR                POINTER ;                                  /* tmp pointer for ALL class qualifier loop*/
     2
     2       Declare
     2       1 CLASSTABLE  GLOBALREF,                                              /* Table of class names & numbers */
     2         2 VECTOR_CT   FIXED BINARY(31),                                     /* Count of longwords following in table */
     2         2 CL_DESCR (0:127),
     2           3 CL_PTR     POINTER,                                             /* Pointer to class cstring */
     2           3 CL_NO      FIXED BINARY(31);                                    /* Class number */
     2
     2       Declare
     2       1 STAT_TABLE     GLOBALREF,                                           /* Table of pointers to str descrs for statistic qua
     2         2 STAT_DESC    (0:STATS-1) POINTER;
     2
     2       Declare
     2       1 PROCD_TABLE    GLOBALREF,                                           /* Table of ptrs to str descrs for PROCESSES display
     2         2 PROCD_DESC   (0:PROCDISPS-1) POINTER;
     2
     2       Declare
     2       I                FIXED BINARY(7),                                     /* Loop control */
     2       J                FIXED BINARY(7),                                     /* Loop control */
     2       QUAL_FOUND       BIT(1) ALIGNED,                                      /* NO => haven't seen a qualifier yet */
     2       CLSNO            FIXED BINARY(31),                                    /* Class number */
     2       QD               CHAR(8) BASED,                                       /* Dummy qualifier string descr */
     2       CLSTR            CHAR(1) BASED;                                       /* Dummy first char of class cstring */
     2
     2       %REPLACE         ITEM_TOK_SIZE           BY 25;                       /* Size of token for item name */
     2
     2       Declare
     2       REQUEST_ITEM_MASK          BIT(16),                                   /* Requested items have bits on */
     2       REQUEST_ITEM_VEC           (0:15) BIT(1) DEFINED(REQUEST_ITEM_MASK);
   : 2                                                                             /* Alias for REQUEST_ITEM_MASK, but bit-addressable
     2
     2       Declare
     2       01 ITEM_VAL STATIC,                                                   /* String descr for value of /ITEM qualifier */
   : 2                                                                             /* Note -- must be STATIC to get INVITEMNM msg */
     2           02 L                   FIXED BINARY(31),                          /* Length */
     2           02 A                   POINTER,                                   /* Address */
     2           02 S                   CHAR(ITEM_TOK_SIZE);                       /* String */
     2
```

```
        2       Declare
        2       IVAL_LEN                        FIXED BINARY(15),                   /* Actual length of item value string */
        2       ITEM_LTAB                       FIXED BINARY(31) BASED,             /* Dummy first longword of item lookup table */
        2       ITEM_KEY                        FIXED BINARY(31);                   /* Item keyword number */
      : 2                                                                           /* Note -- must be STATIC to get error msg */
        2
        2
```

```
    2
  : 2        /*
  : 2        /*      NOTE -- CDBPTR (and its synonym C) has been set up by caller.
  : 2        /*/
    2
  : 2        /*
  : 2        /*      Check for the presence of each of the statistics qualifiers
  : 2        /*      (ALL, CUR, AVE, MIN, MAX). If specified for the non-standard
  : 2        /*      class (PROCESSES), or if more than one specified, log error
  : 2        /*      and return.
  : 2        /*/
    2
    2        QUAL_FOUND = NO;                                               /* Indicate no statistics qualifiers found yet */
    2
    2        DO I = 0 TO STATS-1;                                           /* Loop for each stat qual for this class */
    3
    3        IF CLI$PRESENT(STAT_DESC(I)->QD)                               /* If this stat qual is present, */
    3            THEN IF C->CDB$V_STD = NO                                  /* Check if non-standard class */
    3                      THEN DO;                                         /* Non-STD -- stat quals not allowed */
    4                           CALL MON_ERR(MNR$_QUALINV,,STAT_DESC(I)->QD,CL_PTR(CLSNO)->CLSTR); /* Log an error */
    4                           RETURN(MNR$_QUALINV);                                          /* ... and return with status *
    4                           END;
    3                      ELSE IF QUAL_FOUND = NO                          /* STD class -- If we haven't seen a qual yet, */
    3                                THEN DO;
    4                                     QUAL_FOUND = YES;                 /* Indicate we found one this time */
    4                                     IF CLSNO = ALL_CLSNO              /* If ALL classes  */
    4                                     THEN DO J = 1 TO MAX_CLASS_NO;    /* Loop once for each possible class */
    5                                          IF C->CDB$V_STD = YES        /* If this is a standard class */
    5                                          THEN DO;
    6                                               TEMP_CDBPTR = ADDR(CDBLOCK(J));          /* ... Get CDB addressability */
    6                                               TEMP_CDBPTR->CDB$B_ST = I;               /* ... and move in the requested sta
    6                                               TEMP_CDBPTR->CDB$V_EXPLIC = YES;         /* ... also indicate explicit qualif
    6                                               END;
    5                                          END;                         /* End ALL classes logic */
    4                                     C->CDB$B_ST = I;                  /* Move in the requested stat code */
    4                                     C->CDB$V_EXPLIC = YES;            /* Indicate a class qualifier explicitly specified *
    4                                     END;
    3                                ELSE DO;                               /* We've seen a stat qual for this class already */
    4                                     CALL MON_ERR(MNR$_SQUALERR,,CL_PTR(CLSNO)->CLSTR); /* Log an error */
    4                                     RETURN(MNR$_SQUALERR);                             /* Return with status */
    4                                     END;
    3        END;
    2
```

```
 : 2        /*
 : 2        /*      Check for the presence of each of the PROCESSES display qualifiers
 : 2        /*      (TOPCPU, TOPFAULT, TOPDIO, TOPBIO). If specified for a standard
 : 2        /*      class, or if more than one specified, log error and return.
 : 2        /*/
   2
   2        QUAL_FOUND = NO;                                             /* Indicate no PROCESSES qualifiers found yet */
   2
   2        DO I = 0 TO PROCDISPS-1;                                     /* Loop for each PROCESSES qualifier */
   3
   3        IF PROCD_DESC(I) ^= NULL() &                                 /* If this proc qual is defined ... */
   3           CLISPRESENT(PROCD_DESC(I)->QD)                            /* ... AND it is present, */
   3             THEN IF C->CDB$V_STD = YES                              /* Check if standard class */
   3                     THEN DO;                                        /* STD class -- proc quals not allowed */
   4                          CALL MON_ERR(MNR$_QUALINV,,PROCD_DESC(I)->QD,CL_PTR(CLSNO)->CLSTR); /* Log an error */
   4                          RETURN(MNR$_QUALINV);                                          /* ... and return with status *
   4                          END;
   3                     ELSE IF QUAL_FOUND = NO                         /* Non-STD class -- If we haven't seen a qual yet, *
   3                               THEN DO;
   4                                    QUAL_FOUND = YES;                /* Indicate we found one this time */
   4                                    C->CDB$B_ST = I;                 /* ... and move in the requested display type */
   4                                    END;
   3                               ELSE DO;                              /* We've seen a proc qual for this class already */
   4                                    CALL MON_ERR(MNR$_PDQUALERR,,CL_PTR(CLSNO)->CLSTR); /* Log an error */
   4                                    RETURN(MNR$_PDQUALERR);                         /* Return with status */
   4                                    END;
   3        END;
   2
 :  2        /*
 :  2        /*      Check for the presence of the /ITEM class qualifier. If present,
 :  2        /*      call CLI$GET_VALUE in a loop to obtain the requested item keywords.
 :  2        /*      Then call LIB$LOOKUP_KEY to get numerical bit values for the keywords
 :  2        /*      and set the bits (in CDX$W_IBITS) corresponding to the bit values.
 :  2        /*/
    2
    2        IF CLI$PRESENT(QUAL$L_ITEM)                                    /* If /ITEM qual is present, */
    2            THEN IF C->CDB$V_HOMOG = NO                                /* Check if homogeneous class */
    2                 THEN DO;                                              /* Not homog -- /ITEM qual not allowed */
    3                      CALL MON_ERR(MNR$_QUALINV,,QUAL$L_ITEM,CL_PTR(CLSNO)->CLSTR); /* Log an error */
    3                      RETURN(MNR$_QUALINV);                                      /* ... and return with status */
    3                      END;
    2
    2                 ELSE DO;                                              /* Homog class -- loop picking up items */
    3                      ITEM_VAL.L = LENGTH(ITEM_VAL.S);                 /* Init length longword of descr */
    3                      ITEM_VAL.A = ADDR(ITEM_VAL.S);                   /* Init address longword of descr */
    3                      REQUEST_ITEM_MASK = '0'B;                        /* Turn off all item bits initially */
    3
    3                      DO WHILE(CLI$GET_VALUE(QUAL$L_ITEM,ITEM_VAL,IVAL_LEN)); /* Loop once for each requested item */
    4                          ITEM_VAL.L = IVAL_LEN;                       /* Get actual string length */
    4
    4                          IF ^ LIB$LOOKUP_KEY(ITEM_VAL,C->CDB$A_CDX->CDX$A_ILOOKTAB->ITEM_LTAB,ITEM_KEY)
    4                              THEN DO;                                 /* Get item keyword number */
    5                                   CALL MON_ERR(MNR$_INVITEMNM,,ITEM_VAL);  /* Log error if bad item name */
    5                                   RETURN(MNR$_INVITEMNM);                  /* ... and return with status */
    5                                   END;
    4
    4                              ELSE DO;                                 /* Found value for item keyword */
    5                                   REQUEST_ITEM_VEC(ITEM_KEY) = YES; /* Turn on bit for this item */
    5                                   ITEM_VAL.L = LENGTH(ITEM_VAL.S); /* Restore string len for next loop */
    5                                   END;
    4
    4                      END;                                             /* End of DO WHILE loop */
    3
    3                      IF INDEX(REQUEST_ITEM_MASK,'1'B) ^= 0            /* If any items specified, */
    3                          THEN C->CDB$A_CDX->CDX$W_IBITS = REQUEST_ITEM_MASK; /* ... move them into the CDX */
    3
    3                      END;
    2
```

```
: 2   /*
: 2   /*      The following Begin-End group sets the CDB$V_PERCENT and CDB$V_CPU bits to
: 2   /*      the proper states, depending on the specifications of the /[NO]PERCENT and
: 2   /*      /[NO]CPU qualifiers.
: 2   /*/
  2
  2   BEGIN;
  3   Declare
  3   CLI$PRESENT     EXTERNAL ENTRY(ANY) RETURNS(FIXED BINARY(31)),  /* CLE routine to determine presence of qualifiers *
  3   CLI$_LOCNEG     FIXED BINARY(31)    GLOBALREF VALUE;            /* CLI$PRESENT return status code for "explicitly ne
  3
: 3   /*
: 3   /*      Check for presence of /[NO]PERCENT qualifier
: 3   /*/
  3
  3   CALL = CLI$PRESENT(QUAL$L_PCENT);                               /* Get "present" indicators for /PERCENT */
  3   IF STATUS                                                       /* If present, */
  3       THEN IF C->CDB$V_UNIFORM                                    /* If uniform (/PERCENT allowed for unif only) */
  3                THEN C->CDB$V_PERCENT = YES;                       /* ... then indicate percent stats */
  3                ELSE DO;
  4                     CALL MON_ERR(MNR$_QUALINV,,QUAL$L_PCENT,CL_PTR(CLSNO)->CLSTR); /* ... otherwise, log an error */
  4                     RETURN(MNR$_QUALINV);                                        /* ... and return with status */
  4                     END;
  3       ELSE IF CALL = CLI$_LOCNEG THEN C->CDB$V_PERCENT = NO;      /* If /PERCENT explicitly negated, turn it off */
  3
: 3   /*
: 3   /*      Now check for presence of /[NO]CPU qualifier
: 3   /*/
  3
  3   CALL = CLI$PRESENT(QUAL$L_CPU);                                 /* Get "present" indicators for /CPU */
  3   IF STATUS                                                       /* If present, */
  3       THEN IF CLSNO = MODES_CLSNO                                 /* If MODES class, */
  3                THEN C->CDB$V_CPU = YES;                           /* ... indicate CPU-specific display */
  3                ELSE DO;                                           /* Otherwise, /CPU is not allowed */
  4                     CALL MON_ERR(MNR$_QUALINV,,QUAL$L_CPU,CL_PTR(CLSNO)->CLSTR); /* Log an error */
  4                     RETURN(MNR$_QUALINV);                                      /* ... and return with status */
  4                     END;
  3       ELSE IF CALL = CLI$_LOCNEG THEN C->CDB$V_CPU = NO;          /* If /CPU explicitly negated, turn it off */
  3
  3   END;                                                            /* Terminate Begin-End group */
  2
  2   RETURN(NORMAL);
  2
  2   END GET_CLASS_QUALS;
  1
  1   END GET_CLASSES;
```

```
             MONITOR_CMD: Procedure Returns (Fixed Binary(31));          /* Routine to execute a MONITOR subcommand */
      1
  :   1      /***
  :   1      /*
  :   1      /* FUNCTIONAL DESCRIPTION:
  :   1      /*
  :   1      /*      MONITOR_CMD
  :   1      /*
  :   1      /*      TBS
  :   1      /*
  :   1      /* INPUTS:
  :   1      /*
  :   1      /*      TBS
  :   1      /*
  :   1      /* IMPLICIT INPUTS:
  :   1      /*
  :   1      /*      TBS
  :   1      /*
  :   1      /* OUTPUTS:
  :   1      /*
  :   1      /*      TBS
  :   1      /*
  :   1      /* IMPLICIT OUTPUTS:
  :   1      /*
  :   1      /*      TBS
  :   1      /*
  :   1      /* ROUTINE VALUE:
  :   1      /*
  :   1      /*      TBS
  :   1      /*
  :   1      /* SIDE EFFECTS:
  :   1      /*
  :   1      /*      TBS
  :   1      /*
  :   1      /**/
      1
```

```
 ! 1     /*
 ! 1     /*      +--------------------------------------------------------------------+
 ! 1     /*      |                                                                    |
 ! 1     /*      |                          LOCAL STORAGE                             |
 ! 1     /*      |                                                                    |
 ! 1     /*      +--------------------------------------------------------------------+
 ! 1     /*/
   1
   1     Declare
   1       CALL                  FIXED BINARY(31),                       /* Holds function value (return status) of called ro
   1       STATUS                BIT(1) BASED(ADDR(CALL)),               /* Low-order status bit for called routines */
   1       AT_LEAST_ONE_CLASS    BIT(1) ALIGNED,                         /* YES => user requested at least one class */
   1       IFBPTR                POINTER;                                /* Pointer to Input File Block (IFB) */
   1
   1     %REPLACE                NOT_SUCCESSFUL      BY '0'B;            /* Failing severity code */
   1
 ! 1     /*
 ! 1     /*      +--------------------------------------------------------------------+
 ! 1     /*      |                                                                    |
 ! 1     /*      |                       EXTERNAL REFERENCES                          |
 ! 1     /*      |                                                                    |
 ! 1     /*      +--------------------------------------------------------------------+
 ! 1     /*/
   1
   1     %INCLUDE        MONDEF;                                         /* Monitor utility structure definitions */
   1
   1     Declare
   1     MON_ERR         ENTRY (ANY VALUE, ANY, ANY) OPTIONS(VARIABLE),  /* MONITOR MACRO-32 routine to log synchronous error
   1     MOVE_CLASS_QUALS ENTRY (FIXED BINARY(7))                        /* MACRO-32 routine to move class ... */
   1                      RETURNS(FIXED BINARY(31)),                     /* ... qualifier values for all classes */
   1     GET_CLASSES     ENTRY (BIT(1) ALIGNED) RETURNS(FIXED BINARY(31)), /* Routine to get info on all classes for this req
   1     GET_QUALIFIERS  ENTRY RETURNS(FIXED BINARY(31)),                /* Routine to store qualifier info for this request
   1     EXECUTE_REQUEST ENTRY RETURNS(FIXED BINARY(31)),                /* Routine to execute a single MONITOR request */
   1     MFSUM_REQUEST   ENTRY RETURNS(FIXED BINARY(31));                /* Routine to execute a single MONITOR request with
   1
   1     Declare
   1       MNR$_NOCLASSES FIXED BINARY(31) GLOBALREF VALUE,              /* Message code */
   1       CUR_TO_ACT     FIXED BINARY(31) GLOBALREF VALUE,              /* Code value for MOVE_CLASS_QUALS rtn */
   1       ALL_TO_ACT     FIXED BINARY(31) GLOBALREF VALUE,              /* Code value for MOVE_CLASS_QUALS rtn */
   1       MRBPTR         POINTER GLOBALREF,                             /* Pointer to MRB (Monitor Request Block) */
   1       M              POINTER DEFINED(MRBPTR),                       /* Short-hand synonym for MRBPTR */
   1       CURR_MRBPTR    POINTER GLOBALREF,                             /* Pointer to "current" MRB (Monitor Request Block)
   1       ACT_MRBPTR     POINTER GLOBALREF,                             /* Pointer to "active" MRB (Monitor Request Block) *
   1       FIRST_MON_CMD  BIT(1) ALIGNED GLOBALREF,                      /* YES => first MONITOR (DCL-level) cmd executing */
   1       QUAL_SPECIFIED BIT(1) ALIGNED GLOBALREF,                      /* YES => at least 1 qualifier explicitly specified
   1       PROMPT         BIT(1) ALIGNED GLOBALREF;                      /* YES => prompt user for another subcommand */
   1
```

```
    1      CALL = MOVE_CLASS_QUALS(CUR_TO_ACT);                          /* Move current class qual values to active */
    1      IF STATUS = NOT_SUCCESSFUL THEN RETURN (CALL);                /* Return if bad status */
    1      ACT_MRBPTR->MRB = CURR_MRBPTR->MRB;                           /* Move current MRB to active MRB */
    1      MRBPTR = ACT_MRBPTR;                                          /* Make all MRB refs refer to "active" MRB */
    1      CALL = GET_CLASSES(AT_LEAST_ONE_CLASS);                       /* Get info on requested classes */
    1      IF STATUS = NOT_SUCCESSFUL THEN RETURN (CALL);                /* Return if bad status */
    1      CALL = GET_QUALIFIERS();                                      /* Get the MONITOR command qualifiers */
    1      IF STATUS = NOT_SUCCESSFUL THEN RETURN (CALL);                /* Return if bad status */
    1
    1      IF AT_LEAST_ONE_CLASS                                         /* If at least one class requested, */
    1          THEN
    1              IF M->MRB$V_MFSUM                                     /* If this is a multi-file summary request, */
    1                  THEN CALL = MFSUM_REQUEST();                      /* Call special REQUEST routine for m.f. summary */
    1                  ELSE DO;                                          /* Regular MONITOR request */
    2                      IF M->MRB$A_INPUT ^= NULL()                   /* If it's a playback, */
    2                          THEN DO;
    3                              IFBPTR = M->MRB$A_INPUT;              /* Make MRB$A_INPUT point to a */
    3                              M->MRB$A_INPUT = IFB$A_INPUT;         /* ... file descr instead of an IFB */
    3                              END;
    2
    2                      IF M->MRB$A_SUMMARY ^= NULL()                 /* If summary requested, */
    2                          THEN DO;
    3                              CALL = MOVE_CLASS_QUALS(ALL_TO_ACT);             /* Force ALL stat value to active */
    3                              IF STATUS = NOT_SUCCESSFUL THEN RETURN (CALL); /* Return if bad status */
    3                              END;
    2
    2                      CALL = EXECUTE_REQUEST();                     /* Execute the MONITOR request */
    2                      END;
    1          ELSE                                                      /* Otherwise, no classes requested, */
    1              IF FIRST_MON_CMD = YES & QUAL_SPECIFIED = NO          /* Check for bare MONITOR cmd at DCL */
    1                  THEN PROMPT = YES;                                /* If so, user wants to go interactive */
    1                  ELSE DO;                                          /* MONITOR cmd issued without classes */
    2                      CALL MON_ERR(MNR$_NOCLASSES);                 /* This is an error ... log it */
    2                      RETURN(MNR$_NOCLASSES);                       /* ... and return */
    2                      END;
    1      RETURN (CALL);                                                /* Return with status code */
    1
    1      END MONITOR_CMD;

```

```
         SET_CMD: Procedure Returns (Fixed Binary(31));                /* Routine to execute a SET subcommand */
   1
 : 1     /***
 : 1     /*
 : 1     /* FUNCTIONAL DESCRIPTION:
 : 1     /*
 : 1     /*      SET_CMD
 : 1     /*
 : 1     /*      TBS
 : 1     /*
 : 1     /* INPUTS:
 : 1     /*
 : 1     /*      TBS
 : 1     /*
 : 1     /* IMPLICIT INPUTS:
 : 1     /*
 : 1     /*      TBS
 : 1     /*
 : 1     /* OUTPUTS:
 : 1     /*
 : 1     /*      TBS
 : 1     /*
 : 1     /* IMPLICIT OUTPUTS:
 : 1     /*
 : 1     /*      TBS
 : 1     /*
 : 1     /* ROUTINE VALUE:
 : 1     /*
 : 1     /*      TBS
 : 1     /*
 : 1     /* SIDE EFFECTS:
 : 1     /*
 : 1     /*      TBS
 : 1     /*
 : 1     /**/
   1
```

```
       1
       1     %INCLUDE       MONDEF;                                              /* Monitor utility structure definitions */
       1
       1     %REPLACE                 NOT_SUCCESSFUL         BY '0'B;            /* Failing severity code */
       1
  |    1     /*
  |    1     /*      |------------------------------------------------------------+
  |    1     /*      |                                                            |
  |    1     /*      |                EXTERNAL STORAGE  DEFINITIONS               |
  |    1     /*      |                                                            |
  |    1     /*      |------------------------------------------------------------+
  |    1     /*/
       1
       1     Declare
       1     INP_PTR_VOL     POINTER GLOBALDEF        INIT(NULL()),              /* Pointer to volatile /INPUT file-spec */
       1     DISP_PTR_VOL    POINTER GLOBALDEF        INIT(NULL()),              /* Pointer to volatile /DISPLAY file-spec */
       1     REC_PTR_VOL     POINTER GLOBALDEF        INIT(NULL()),              /* Pointer to volatile /RECORD file-spec */
       1     SUMM_PTR_VOL    POINTER GLOBALDEF        INIT(NULL()),              /* Pointer to volatile /SUMMARY file-spec */
       1     COMM_PTR_VOL    POINTER GLOBALDEF        INIT(NULL()),              /* Pointer to volatile /COMMENT string */
       1     IFB_TAB_VOL     POINTER GLOBALDEF        INIT(NULL());              /* Pointer to volatile IFB_TABLE */
       1
       1     Declare
       1     INP_PTR_SWAP    BIT(1) ALIGNED GLOBALDEF,                           /* YES => swap INP_PTR_VOL and INP_PTR_PERM */
       1     DISP_PTR_SWAP   BIT(1) ALIGNED GLOBALDEF,                           /* YES => swap DISP_PTR_VOL and DISP_PTR_PERM */
       1     REC_PTR_SWAP    BIT(1) ALIGNED GLOBALDEF,                           /* YES => swap REC_PTR_VOL and REC_PTR_PERM */
       1     SUMM_PTR_SWAP   BIT(1) ALIGNED GLOBALDEF,                           /* YES => swap SUMM_PTR_VOL and SUMM_PTR_PERM */
       1     COMM_PTR_SWAP   BIT(1) ALIGNED GLOBALDEF,                           /* YES => swap COMM_PTR_VOL and COMM_PTR_PERM */
       1     IFB_TAB_SWAP    BIT(1) ALIGNED GLOBALDEF;                           /* YES => swap IFB_TAB_VOL and IFB_TAB_PERM */
       1
  |    1     /*
  |    1     /*      |------------------------------------------------------------+
  |    1     /*      |                                                            |
  |    1     /*      |                    EXTERNAL REFERENCES                     |
  |    1     /*      |                                                            |
  |    1     /*      |------------------------------------------------------------+
  |    1     /*/
       1
       1     Declare
       1     MOVE_CLASS_QUALS ENTRY (FIXED BINARY(7))                            /* MACRO-32 routine to move class ... */
       1                      RETURNS(FIXED BINARY(31)),                         /* ... qualifier values for all classes */
       1     GET_CLASSES     ENTRY (BIT(1) ALIGNED) RETURNS(FIXED BINARY(31)), /* Routine to get info on all classes for this req
       1     GET_QUALIFIERS  ENTRY RETURNS(FIXED BINARY(31));                    /* Routine to store qualifier info for this request
       1
       1     Declare
       1       CUR_TO_ACT    FIXED BINARY(31) GLOBALREF VALUE,                   /* Code value for MOVE_CLASS_QUALS rtn */
       1       ACT_TO_CUR    FIXED BINARY(31) GLOBALREF VALUE,                   /* Code value for MOVE_CLASS_QUALS rtn */
       1       MRBPTR        POINTER GLOBALREF,                                  /* Pointer to MRB (Monitor Request Block) */
       1       CURR_MRBPTR   POINTER GLOBALREF,                                  /* Pointer to "current" MRB (Monitor Request Block)
       1       TEMP_MRBPTR   POINTER GLOBALREF;                                  /* Pointer to "temp" MRB (Monitor Request Block) */
       1
  |    1     /*
  |    1     /*      |------------------------------------------------------------+
  |    1     /*      |                                                            |
  |    1     /*      |                       LOCAL STORAGE                        |
  |    1     /*      |                                                            |
  |    1     /*      |------------------------------------------------------------+
   1         /*/
   1
   1         Declare
   1         INP_PTR_PERM     POINTER STATIC            INIT(NULL()),           /* Pointer to permanent /INPUT file-spec */
   1         DISP_PTR_PERM    POINTER STATIC            INIT(NULL()),           /* Pointer to permanent /DISPLAY file-spec */
   1         REC_PTR_PERM     POINTER STATIC            INIT(NULL()),           /* Pointer to permanent /RECORD file-spec */
   1         SUMM_PTR_PERM    POINTER STATIC            INIT(NULL()),           /* Pointer to permanent /SUMMARY file-spec */
   1         COMM_PTR_PERM    POINTER STATIC            INIT(NULL()),           /* Pointer to permanent /COMMENT string */
   1         IFB_TAB_PERM     POINTER STATIC            INIT(NULL());           /* Pointer to permanent IFB_TABLE */
   1
   1         Declare
   1           CALL                   FIXED BINARY(31),                          /* Holds function value (return status) of called ro
   1           STATUS                 BIT(1) BASED(ADDR(CALL)),                  /* Low-order status bit for called routines */
   1           TEMP                   POINTER,                                   /* Temporary ''scratch'' area */
   1           AT_LEAST_ONE_CLASS     BIT(1) ALIGNED;                            /* YES => user requested at least one class */
   1
   1
   1        INP_PTR_SWAP = NO;                                                  /* Ind no swap needed for 2 ptrs to /INPUT file-spec
   1        DISP_PTR_SWAP = NO;                                                 /* Ind no swap needed for 2 ptrs to /DISPLAY file-sp
   1        REC_PTR_SWAP = NO;                                                  /* Ind no swap needed for 2 ptrs to /RECORD file-spe
   1        SUMM_PTR_SWAP = NO;                                                 /* Ind no swap needed for 2 ptrs to /SUMMARY file-sp
   1        COMM_PTR_SWAP = NO;                                                 /* Ind no swap needed for 2 ptrs to /COMMENT string
   1        IFB_TAB_SWAP = NO;                                                  /* Ind no swap needed for 2 ptrs to IFB_TABLE */
   1        TEMP_MRBPTR->MRB = CURR_MRBPTR->MRB;                                /* Move "current" MRB to "temp" MRB */
   1        MRBPTR = TEMP_MRBPTR;                                               /* Make all MRB refs refer to "temp" MRB */
   1        CALL = GET_QUALIFIERS();                                            /* Get SET qualifiers */
   1        IF STATUS = NOT_SUCCESSFUL THEN RETURN (CALL);                      /* Return if bad status */
   1
   1        CALL = MOVE_CLASS_QUALS(CUR_TO_ACT);                                /* Move current class qual values to active */
   1        IF STATUS = NOT_SUCCESSFUL THEN RETURN (CALL);                      /* Return if bad status */
   1
   1        CALL = GET_CLASSES(AT_LEAST_ONE_CLASS);                             /* Get info on requested classes */
   1
   1        IF STATUS                                                           /* If successful, */
   1            THEN DO;
   2                 TEMP = TEMP_MRBPTR;                                        /* Swap "current" and ... */
   2                 TEMP_MRBPTR = CURR_MRBPTR;                                 /* ... "temp" MRB pointers */
   2                 CURR_MRBPTR = TEMP;                                        /* ..... */
   2
   2                 IF INP_PTR_SWAP = YES                                      /* If pointer swap required, */
   2                     THEN DO;
   3                          TEMP = INP_PTR_VOL;                               /* Swap volatile and ... */
   3                          INP_PTR_VOL = INP_PTR_PERM;                       /* ... permanent /INPUT file-spec */
   3                          INP_PTR_PERM = TEMP;                              /* ... pointers */
   3                          END;
   2
   2                 IF DISP_PTR_SWAP = YES                                     /* If pointer swap required, */
   2                     THEN DO;
   3                          TEMP = DISP_PTR_VOL;                              /* Swap volatile and ... */
   3                          DISP_PTR_VOL = DISP_PTR_PERM;                     /* ... permanent /DISPLAY file-spec */
   3                          DISP_PTR_PERM = TEMP;                             /* ... pointers */
   3                          END;
   2
   2                 IF REC_PTR_SWAP = YES                                      /* If pointer swap required, */
   2                     THEN DO;
   3                          TEMP = REC_PTR_VOL;                               /* Swap volatile and ... */
   3                          REC_PTR_VOL = REC_PTR_PERM;                       /* ... permanent /RECORD file-spec */
   3                          REC_PTR_PERM = TEMP;                              /* ... pointers */
   3                          END;
   2
   2                 IF SUMM_PTR_SWAP = YES                                     /* If pointer swap required, */
   2                     THEN DO;
   3                          TEMP = SUMM_PTR_VOL;                              /* Swap volatile and ... */
   3                          SUMM_PTR_VOL = SUMM_PTR_PERM;                     /* ... permanent /SUMMARY file-spec */
   3                          SUMM_PTR_PERM = TEMP;                             /* ... pointers */
   3                          END;
   2
   2                 IF COMM_PTR_SWAP = YES                                     /* If pointer swap required, */
   2                     THEN DO;
   3                          TEMP = COMM_PTR_VOL;                              /* Swap volatile and ... */
   3                          COMM_PTR_VOL = COMM_PTR_PERM;                     /* ... permanent /COMMENT string */
   3                          COMM_PTR_PERM = TEMP;                             /* ... pointers */
   3                          END;
```

```
2
2                  IF IFB_TAB_SWAP = YES                                  /* If pointer swap required, */
2                      THEN DO;
3                           TEMP = IFB_TAB_VOL;                           /* Swap volatile and ... */
3                           IFB_TAB_VOL = IFB_TAB_PERM;                   /* ... permanent IFB_TABLE */
3                           IFB_TAB_PERM = TEMP;                          /* ... pointers */
3                           END;
2
2                  CALL = MOVE_CLASS_QUALS(ACT_TO_CUR);                   /* Move active class qual values back to current */
2                  IF STATUS = NOT_SUCCESSFUL THEN RETURN (CALL);         /* Return if bad status */
2
2                  END;
1
1       RETURN (CALL);                                                    /* Return with status code */
1
1       END SET_CMD;

```

```
        INIT_CMD: Procedure Returns (Fixed Binary(31));                      /* Routine to execute an INITIALIZE subcommand */
  1
  1     /*++
  1     /*
  1     /* FUNCTIONAL DESCRIPTION:
  1     /*
  1     /*      INIT_CMD
  1     /*
  1     /*      TBS
  1     /*
  1     /* INPUTS:
  1     /*
  1     /*      TBS
  1     /*
  1     /* IMPLICIT INPUTS:
  1     /*
  1     /*      TBS
  1     /*
  1     /* OUTPUTS:
  1     /*
  1     /*      TBS
  1     /*
  1     /* IMPLICIT OUTPUTS:
  1     /*
  1     /*      TBS
  1     /*
  1     /* ROUTINE VALUE:
  1     /*
  1     /*      TBS
  1     /*
  1     /* SIDE EFFECTS:
  1     /*
  1     /*      TBS
  1     /*
  1     /*/
  1
```

```
 ¦  1       /*
 ¦  1       /*       |------------------------------------------------------------------+
 ¦  1       /*       |                                                                  |
 ¦  1       /*       |                           LOCAL STORAGE                          |
 ¦  1       /*       |                                                                  |
 ¦  1       /*       |------------------------------------------------------------------+
 ¦  1       /*/
    1
    1       Declare
    1         CALL                 FIXED BINARY(31);                    /* Holds function value (return status) of called ro
    1
    1
 ¦  1       /*
 ¦  1       /*       |------------------------------------------------------------------+
 ¦  1       /*       |                                                                  |
 ¦  1       /*       |                       EXTERNAL REFERENCES                        |
 ¦  1       /*       |                                                                  |
 ¦  1       /*       |------------------------------------------------------------------+
 ¦  1       /*/
    1
    1       %INCLUDE        MONDEF;                                     /* Monitor utility structure definitions */
    1
    1       Declare
    1       DEF_TO_CUR      FIXED BINARY(31) GLOBALREF VALUE;           /* Code value for MOVE_CLASS_QUALS rtn */
    1
    1       Declare
    1       MOVE_CLASS_QUALS           ENTRY (FIXED BINARY(7))          /* MACRO-32 routine to move class ... */
    1                                  RETURNS(FIXED BINARY(31));       /* ... qualifier values for all classes */
    1
    1       Declare
    1         CURR_MRBPTR   POINTER GLOBALREF,                          /* Pointer to "current" MRB (Monitor Request Block)
    1         DEF_MRBPTR    POINTER GLOBALREF;                          /* Pointer to "default" MRB (Monitor Request Block)
    1
```

```
   1        CURR_MRBPTR->MRB = DEF_MRBPTR->MRB;                              /* Move default MRB to current MRB */
   1        CALL = MOVE_CLASS_QUALS(DEF_TO_CUR);                             /* ... and default class qual values to current */
   1
   1        RETURN (CALL);                                                   /* Return with status code from MOVE_CLASS_QUALS */
   1
   1        END INIT_CMD;

```

```
HELP_CMD: Procedure Returns (Fixed Binary(31));                 /* Routine to execute a HELP subcommand */

/***
/*
/* FUNCTIONAL DESCRIPTION:
/*
/*      HELP_CMD
/*
/*      TBS
/*
/* INPUTS:
/*
/*      TBS
/*
/* IMPLICIT INPUTS:
/*
/*      TBS
/*
/* OUTPUTS:
/*
/*      TBS
/*
/* IMPLICIT OUTPUTS:
/*
/*      TBS
/*
/* ROUTINE VALUE:
/*
/*      TBS
/*
/* SIDE EFFECTS:
/*
/*      TBS
/*
/**/
```

```
%INCLUDE               $HLPDEF;                                  /* HELP flag definitions */
%REPLACE               NOT_SUCCESSFUL          BY '0'B;          /* Failing severity code */
%REPLACE               MON_HELPLIB             BY 'MNRHELP';     /* Log name for MONITOR subcommand help library */
%REPLACE               HELP_PARM               BY 'HELP_KEYS';   /* Name of parameter for HELP command (rest of line)

/*
/*        +-------------------------------------------------------------------------+
/*        |                                                                         |
/*        |                              LOCAL STORAGE                              |
/*        |                                                                         |
/*        +-------------------------------------------------------------------------+
/*/
Declare
   CALL                FIXED BINARY(31),                          /* Holds function value (return status) of called ro
   STATUS              BIT(1) BASED(ADDR(CALL));                  /* Low-order status bit for called routines */

/*
/*        +-------------------------------------------------------------------------+
/*        |                                                                         |
/*        |                          EXTERNAL REFERENCES                            |
/*        |                                                                         |
/*        +-------------------------------------------------------------------------+
/*/

Declare
CLI$GET_VALUE     ENTRY (CHAR(*), ANY)                            /* CLE routine to get qualifier values */
                  RETURNS(BIT(1)),
LBR$OUTPUT_HELP   ENTRY (ENTRY VALUE, ANY, ANY, CHAR(*), FIXED BINARY(31), ENTRY VALUE)
                  OPTIONS(VARIABLE) RETURNS(FIXED BINARY(31)),    /* Rtn to output help info with prompting */
LIB$PUT_OUTPUT    ENTRY,                                          /* Routine to write a line to terminal */
LIB$GET_INPUT     ENTRY,                                          /* Routine to read terminal */
MON_ERR           ENTRY (ANY VALUE, ANY, ANY) OPTIONS(VARIABLE),
                                                                  /* Routine to log synchronous errors */
MNR$_HELPERR      FIXED BINARY(31) GLOBALREF VALUE;               /* Error message code */

Declare
NORMAL            FIXED BINARY(31) GLOBALREF,                     /* MONITOR normal return status */
CURR_ERRCODE      FIXED BINARY(31) GLOBALREF;                     /* MONITOR error status code currently expected */

Declare
1 DYN_STRING      GLOBALREF,                                      /* Dynamic string descriptor */
  2 L             FIXED BINARY(15),                               /* Length */
  2 TC            CHAR(2),                                        /* Type and Class */
  2 A             POINTER;                                        /* Address */
```

```
   1
   1       CURR_ERRCODE = MNR$_HELPERR;                                      /* Set MONITOR code for signaled errors */
   1       IF ^ CLI$GET_VALUE(HELP_PARM,DYN_STRING)                          /* Get all command line after "HELP" verb */
 ! 1                                                                         /* ... then ship it off to LBR$OUTPUT_HELP */
   1           THEN DYN_STRING.L = 0;                                        /* If rest of line null, make string null */
   1
   1       CALL = LBR$OUTPUT_HELP(LIB$PUT_OUTPUT,,DYN_STRING,MON_HELPLIB,
   1                              HLP$M_PROMPT+HLP$M_PROCESS+HLP$M_GROUP+HLP$M_SYSTEM+HLP$M_HELP,
   1                              LIB$GET_INPUT);                            /* Call standard HELP output routine */
   1
   1       IF STATUS = NOT_SUCCESSFUL                                        /* If failed, */
   1           THEN DO;
   2                CALL MON_ERR(MNR$_HELPERR,CALL);                         /* Log the error ... */
   2                RETURN(MNR$_HELPERR);                                    /* ... and return with status */
   2                END;
   1
   1       RETURN (NORMAL);                                                  /* Return with success code */
   1
   1       END HELP_CMD;
```

```
        EXIT_CMD: Procedure Returns (Fixed Binary(31));                    /* Routine to execute an EXIT subcommand */
   1
:  1    /***
:  1    /*
:  1    /* FUNCTIONAL DESCRIPTION:
:  1    /*
:  1    /*      EXIT_CMD
:  1    /*
:  1    /*      TBS
:  1    /*
:  1    /* INPUTS:
:  1    /*
:  1    /*      TBS
:  1    /*
:  1    /* IMPLICIT INPUTS:
:  1    /*
:  1    /*      TBS
:  1    /*
:  1    /* OUTPUTS:
:  1    /*
:  1    /*      TBS
:  1    /*
:  1    /* IMPLICIT OUTPUTS:
:  1    /*
:  1    /*      TBS
:  1    /*
:  1    /* ROUTINE VALUE:
:  1    /*
:  1    /*      TBS
:  1    /*
:  1    /* SIDE EFFECTS:
:  1    /*
:  1    /*      TBS
:  1    /*
:  1    /*/
   1
```

```
 |  1        /*
 |  1        /*      +----------------------------------------------------------------+
 |  1        /*      |                                                                |
 |  1        /*      |                         LOCAL STORAGE                          |
 |  1        /*      |                                                                |
 |  1        /*      +----------------------------------------------------------------+
 |  1        /*/
    1
    1
    1
 |  1        /*
 |  1        /*      +----------------------------------------------------------------+
 |  1        /*      |                                                                |
 |  1        /*      |                      EXTERNAL REFERENCES                       |
 |  1        /*      |                                                                |
 |  1        /*      +----------------------------------------------------------------+
 |  1        /*/
    1
    1        Declare
    1          PROMPT        BIT(1) ALIGNED  GLOBALREF,                        /* YES => prompt user for another subcommand */
    1          NORMAL        FIXED BINARY(31) GLOBALREF;                        /* MONITOR normal return status */
    1
```

```
     1      PROMPT = NO;                                             /* Indicate no more prompting for subcommands */
     1
     1      RETURN (NORMAL);                                         /* Return with status code */
     1
     1      END EXIT_CMD;


```

```
EXECUTE_CMD: Procedure Returns (Fixed Binary(31));              /* Routine to execute a MONITOR command file */

/*++
/*
/* FUNCTIONAL DESCRIPTION:
/*
/*      EXECUTE_CMD
/*
/*      This is the routine invoked by CLI$DISPATCH when the EXECUTE command is given to MONITOR
/*      at the interactive level.  EXECUTE_CMD opens the execute command file, sets the bit flag EXECUTE,
/*      and returns to the main interactive loop in MONMAIN (which in turn calls NEXT_EXECUTE_COMMAND).
/*      If there is an error opening the file, the error is reported and control returns
/*      to the interactive level.
/*
/* INPUTS:
/*
/*      Execute filename - this is the file that will be opened so NEXT_EXECUTE_COMMAND can read monitor
/*          commands.  The filename is obtained by using CLI$GET_VALUE with the CLD label "EXEC_FILE".
/*
/* IMPLICIT INPUTS:
/*
/*      EXECUTE - a bet set to tell the main loop to call NEXT_EXECUTE_COMMAND instead of EXECUTE_COMMAND.
/*          NEXET_EXECUTE_COMMAND reads MONITOR commands from the opened file instead of the terminal.
/*
/* OUTPUTS:
/*
/*      None.
/*
/* IMPLICIT OUTPUTS:
/*
/*      COMMAND_FILE - the file reference associated with the execute command file.  This is also used by
/*          NEXT_EXECUTE_COMMAND to read the file.
/*
/* ROUTINE VALUE:
/*
/*      MNR$_ERREXECOM  - EXECUTE subcommand cannot be used in an execute command file.
/*      MNR$_ERREXEFIL  - CLI$GET_VALUE failed to return the execute file name.
/*      MNR$_ERREXEOPN  - Open failure using the execute file name.
/*      SS$_NORMAL      - Success.
/*
/* SIDE EFFECTS:
/*
/*      None.
/*
/*/
```

```
 !  1        /*
 !  1        /*      +------------------------------------------------------------------------------+
 !  1        /*      !                                                                              !
 !  1        /*      !                              REQUIRE FILE                                    !
 !  1        /*      !                                                                              !
 !  1        /*      +------------------------------------------------------------------------------+
 !  1        /*/
    1
    1        %INCLUDE        MONDEF;                                          /* Monitor utility structure definitions */
    1
    1        %REPLACE        FILE_SPEC_SIZE          BY 128;                  /* Max file spec size */
    1        %REPLACE        NOT_SUCCESSFUL          BY '0'B;                 /* Failing severity code */
    1
    1
 !  1        /*
 !  1        /*      +------------------------------------------------------------------------------+
 !  1        /*      !                                                                              !
 !  1        /*      !                   GLOBAL STORAGE DEFINITIONS                                 !
 !  1        /*      !                                                                              !
 !  1        /*      +------------------------------------------------------------------------------+
 !  1        /*/
    1
    1        Declare
    1            COMMAND_FILE        FILE RECORD GLOBALDEF;                   /* Execute command file */
    1
 !  1        /*
 !  1        /*      +------------------------------------------------------------------------------+
 !  1        /*      !                                                                              !
 !  1        /*      !                         EXTERNAL REFERENCES                                  !
 !  1        /*      !                                                                              !
 !  1        /*      +------------------------------------------------------------------------------+
 !  1        /*/
    1
    1        Declare
    1            CLI$GET_VALUE       ENTRY(ANY, ANY)                          /* CLE routine to get qualifier values */
    1                                RETURNS(FIXED BINARY(31)),
    1            MON_ERR             ENTRY (ANY VALUE, ANY, ANY)              /* Routine to log synchronous errors */
    1                                OPTIONS(VARIABLE);
    1        Declare
    1            CURR_ERRCODE        FIXED BINARY(31) GLOBALREF,              /* MONITOR error status code currently expected */
    1            EXECUTE             BIT(1) ALIGNED   GLOBALREF,              /* YES => read another command from the execute file
    1            NORMAL              FIXED BINARY(31) GLOBALREF;              /* MONITOR normal return status */
    1
    1        Declare
    1            MNR$_ERREXECOM      FIXED BINARY(31) GLOBALREF VALUE,        /* Error message code */
    1            MNR$_ERREXEFIL      FIXED BINARY(31) GLOBALREF VALUE,        /* Error message code */
    1            MNR$_ERREXEOPN      FIXED BINARY(31) GLOBALREF VALUE;        /* Error message code */
    1
    1
 !  1        /*
 !  1        /*      +------------------------------------------------------------------------------+
 !  1        /*      !                                                                              !
 !  1        /*      !                              LOCAL STORAGE                                   !
 !  1        /*      !                                                                              !
 !  1        /*      +------------------------------------------------------------------------------+
 !  1        /*/
```

```
      1
      1        Declare
      1             CALL                  FIXED BINARY(31),                       /* Holds function value (return status) of called ro
      1             STATUS                BIT(1) BASED(ADDR(CALL));               /* Low-order status bit for called routines */
      1
      1        Declare
      1             1 EXEC_FILE_PARM,
      1               2 L                 FIXED BINARY (31),
      1               2 A                 POINTER,
      1
      1             1 EXEC_FILE_VAL,
      1               2 L                 FIXED BINARY (31),
      1               2 A                 POINTER,
      1
      1             EXEC_FILE_NAME        CHAR(9)  STATIC INIT('EXEC_FILE'),
      1             EXEC_FILE_STR         CHAR(FILE_SPEC_SIZE) STATIC INIT('MONITOR.MON');
      1
      1
      1        ON UNDEFINEDFILE (COMMAND_FILE) GOTO OPEN_ERROR;                   /* Set up the UNDEFINEDFILE condition */
      1
      1
      1        IF EXECUTE = YES                                                   /* Make sure there is only 1 level of execute comman
      1             THEN DO;
      2                  CALL MON_ERR(MNR$_ERREXECOM);                            /* Log the error... */
      2                  RETURN(MNR$_ERREXECOM);                                  /* Return to processing the execute command file */
      2             END;
      1
      1        EXECUTE = YES;                                                     /* Let everyone know command input is coming from a
      1        EXEC_FILE_PARM.L = LENGTH(EXEC_FILE_NAME);                         /* Set the length of the EXEC_FILE descriptor */
      1        EXEC_FILE_PARM.A = ADDR(EXEC_FILE_NAME);                           /* Set the address of the EXEC_FILE string */
      1        EXEC_FILE_VAL.L =  LENGTH(EXEC_FILE_STR);                          /* Set the length of the execute filename descriptor
      1        EXEC_FILE_VAL.A =  ADDR(EXEC_FILE_STR);                            /* Set the address of the execute filename string */
      1        CURR_ERRCODE = MNR$_ERREXEFIL;                                     /* Set MONITOR code for signaled errors */
      1        CALL = CLI$GET_VALUE(EXEC_FILE_PARM,EXEC_FILE_VAL);                /* Get EXECUTE file name */
      1        IF STATUS = NOT_SUCCESSFUL                                         /* If failed, */
      1             THEN DO;
      2                  EXECUTE = NO;                                            /* Error, no longer doing an EXECUTE subcommand */
      2                  CALL MON_ERR(MNR$_ERREXEFIL,CALL);                       /* Log the error ... */
      2                  RETURN(MNR$_ERREXEFIL);                                  /* ... and return with status */
      2                  END;
      1
      1        OPEN FILE(COMMAND_FILE) INPUT SEQUENTIAL TITLE(EXEC_FILE_STR)      /* Open the execute command file */
      1             ENVIRONMENT(DEFAULT_FILE_NAME('.MON'));                       /*  with .MON for default file type */
      1
      1        CURR_ERRCODE = 0;                                                  /* Set MONITOR code for signaled errors */
      1        RETURN (NORMAL);                                                   /* Return with success code */
      1
      1        OPEN_ERROR:
      1             EXECUTE = NO;                                                 /* Indicate no more from the execute file */
      1             CURR_ERRCODE = 0;                                             /* Reset to default MONITOR code in case subcommand
      1             CALL MON_ERR(MNR$_ERREXEOPN);                                 /* Log the error ... */
      1             RETURN(MNR$_ERREXEOPN);                                       /* ... and return with status */
      1
      1        END EXECUTE_CMD;

COMMAND LINE
```

```
------- ----

PLI/LIS=LIS$:MONMAIN/OBJ=OBJ$:MONMAIN MSRC$:MONMAIN+LIB$:MONLIB/LIB
```

B 1

PR
VO

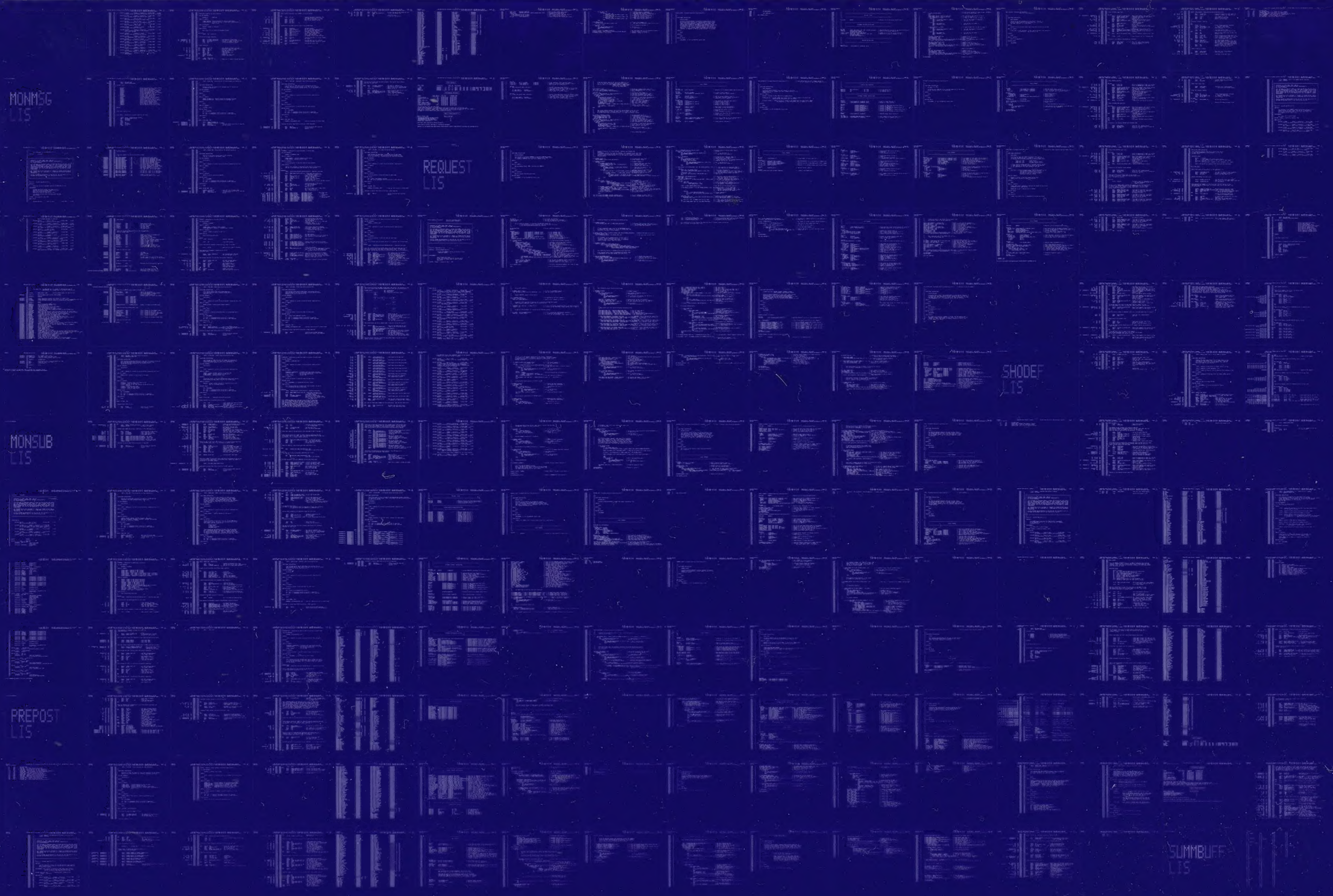
0241 AH-BT13A-SE
VAX/VMS V4.0

DIGITAL EQUIPMENT CORPORATION
CONFIDENTIAL AND PROPRIETARY

MONMAIN
LIS

0242 AH-BT13A-SE
VAX/VMS V4.0

DIGITAL EQUIPMENT CORPORATION
CONFIDENTIAL AND PROPRIETARY


MONMSG
LIS

REQUEST
LIS

SHODEF
LIS

MONSUB
LIS

PREPOST
LIS

SUMMBUFF
LIS